380Z
and
LINK 480Z

FIRMWARE
REFERENCE
MANUAL

FIRMWARE REFERENCE MANUAL

PN 10971 Revision 1

CP/M, CP/NET, and CP/NOS are the names of proprietary products mentioned in this publication, and they are trademarks of Digital Research Inc. WordStar is a trademark of MicroPro International Corporation. Z80 is a trademark of Zilog corporation.

Because of our policy of continuous improvement in our products and services, we may make changes without notice. We have tried to keep the information in this publication completely accurate; however, we cannot be held responsible for the consequences of any errors or omissions.

Customers comments are of great value to us in improving the quality of our microcomputer systems, publications, and services. If you would like to make any comments, please use the reply-paid form provided for this purpose at the back of this manual.

****

Written and published by Research Machines Limited using the WordStar word-processing package running on a Research Machines 380Z-D microcomputer. Printed by Hazel Press, Abbey Trading Estate, Wembley, Middlesex HA0 1YT.

Research Machines Limited
Mill Street
Oxford
OX2 0BW

Phone: Oxford (0865) 249 866

PREFACE

This manual is intended for application programmers and systems programmers who wish to use the operating system facilities provided by the COS firmware in 380Z computers and the ROS firmware in 480Z computers.

The manual assumes that you are familiar with the operation of your computer (described in the relevant Users Guide) and how to program under CP/M (described in CP/M and CP/NET Programmer's Guide, PN 12084).

Most people who program the 380Z and 480Z will use CP/M. However, you may wish to resort to the facilities described in this book when:

- writing fault-tolerant software (CP/M does not return much diagnostic information to your programs)
- writing programs that handle cassette tape
- adapting programs designed to run on other manufacturers' equipment to run on Research Machines computers
- writing time-critical programs where CP/M facilities are not fast enough
- taking advantage of some of the special features of the 380Z and 480Z (that are not possible under CP/M).

This book is mainly concerned with firmware, but, in some instances, the boundary between firmware and hardware is not precise. Hardware is described in the following books:

380Z Disc System Information File, PN 10930

Link 480Z Information File, PN 10939

Related Publications

Related topics are covered in the following publications:

The Zilog Z80 instruction set	-	MOSTEK Assembly Language Programming Manual, PN 11069
Assembly Language programming and the use of the Front Panel	-	Machine Language Programming Guide for 380Z and 480Z, PN 11068
Research Machines ZASM assembler	-	ZASM Assembler for Disc and Network Systems, PN 11066
Writing programs under the control of CP/M or CP/NET	-	CP/M and CP/NET Programmer's Guide, PN 12084

Note

This manual replaces the following books:

- COS 3.4 Reference Manual, PN 10949
- COS 4.0 Differences from COS 3.4 Reference Notes, PN 10958
- LINK 480Z Resident Operating System 1.0 Reference Manual, PN 10965.

CONTENTS

Contents

Contents

CHAPTER 1

INTRODUCTION

This chapter outlines the scope and contents of this manual.

Your computer system has several 'layers' of operation, as illustrated in figure 1.1.

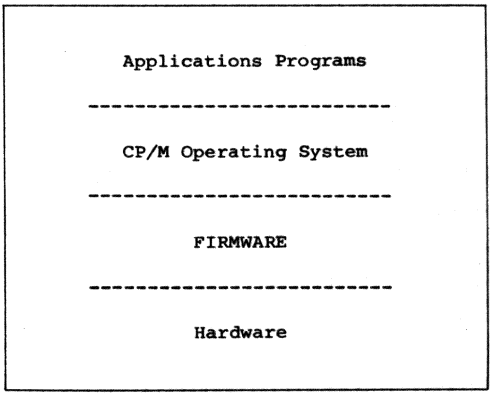


Figure 1.1 Layers of operation

The lowest layer is the hardware, the electronics of your computer. The hardware is driven by the firmware that carries out frequently-repeated actions, such as keyboard entry and output to the screen.

Above the firmware, in a disc-based system, is the CP/M operating system software that controls disc operations. The top layer is the application program of your choice.

This manual is primarily concerned with the firmware layer, and how the facilities it provides can be used when writing assembly language programs.

The first section defines the level of knowledge that is assumed of the reader. The next two sections define what firmware is, and how an important firmware facility, the EMT instruction, is used. Other firmware facilities are discussed and, finally, the conventions used in this manual are described.

SCOPE OF THE MANUAL

We assume that you are familiar with assembler programming. If you are not, the Research Machines manual: Machine Language Programming Guide for 380Z and 480Z provides an introduction to Z80 programming. There are several instructional books on the market; contact your local bookseller.

Also, it is taken that you know how to:

- produce a text file of mnemonics using a text editor package such as TXED
- use an assembler, such as ZASM, to assemble the text file into machine code
- produce a .REL file (relocatable program) or a .HEX file (Intel hexadecimal Format)
- use appropriate utilities such as LOAD and DDT (CPM), or a linker, to produce an executable .COM file.

Research Machines manuals on these topics are listed below:

TXED Text Editor for Disc System and Network Users Reference Manual, PN 11059

ZASM Assembler for Disc and Network Systems, PN 11066

CP/M Operating System Version 2.2D CP/M and CP/NET Programmer's Guide, PN 12084

Further information can be obtained from Digital Research manuals, obtainable from Research Machines.

WHAT IS FIRMWARE?

Like all computers, the 380Z and 480Z have machine-language programs permanently stored in read-only memory. These programs are the bridge between the computer hardware and the software, and are termed 'firmware'.

Firmware controls the interfaces between peripherals, software, memory and yourself. For example, when your program outputs text to the screen or printer, the operation is controlled by the resident firmware.

Firmware controls the internal operations of the computer, supports CP/M (the World's leading disc-based operating system), and provides a number of additional features not found on other computers:

- it performs certain functions from commands that you enter at the keyboard
- you can write and check low-level (machine language) programs using a facility known as the Front Panel.

- you can write low-level programs that use firmware facilities. The EMT instruction is used to do this.

- you can link in your own low-level routines.

In 380Z machines, the firmware is called the "Central Operating System" (COS); in 480Z machines it is known as the "Resident Operating System" (ROS). There have been several versions of COS and ROS as upgrades have been made to improve performance, but all versions have been very similar in design to ensure as much program compatibility as possible. In the case of COS, versions earlier than COS 3.0 are not covered in this manual.

Identifying Your Firmware Version

In subsequent chapters, we describe the firmware in detail. Some of the facilities may not be available in your firmware version. To make it easier for you, we describe the differences between firmware versions at the start of each chapter.

When you switch on or reset your 380Z or 480Z, the firmware displays its name and version number on the screen (the start-up message). Table 1.1 summarizes the firmware versions for the 380Z, and also shows the start-up message for each version.

Table 1.1 380Z Firmware version and start-up messages

Version	Revision	System Type	Start-up Message
COS 3.0	-	C	COS 3.0/C
		M	COS 3.0/M
		F	COS 3.0/F
COS 3.4	C	C	COS 3.4 C/C
	D		COS 3.4 D/C
	C	M	COS 3.4 C/M
	D		COS 3.4 D/M
	C	F	COS 3.4 C/F
	E		COS 3.4 E/F
COS 4.0	A	M	COS 4.0 A/M
	B		COS 4.0 B/M
	-	F	COS 4.0/F
	A		COS 4.0 A/F
COS 4.2	A	-	COS 4.2 A

In the table, a revision covers minor changes that do not affect the user. System type in COS 3.0, 3.4 and 4.0 refers to the type of external storage: cassette tape (C); 5.25-inch, single-density disc (M); and 8-inch, single-density disc (F). In COS 4.2, there is no system type, but systems can have either 5.25-inch or 8-inch double-density discs.

If the firmware version of your system is COS 3.0 or earlier, you are strongly recommended to consult Research Machines about updating to a more recent version; more recent applications software requires the features of COS 3.4 and later.

Table 1.2 summarizes firmware versions for the 480Z, with start-up messages. There are no system-type designations, but ROS 1.0 and ROS 1.1 support cassette tape external storage while ROS 1.2 and 2.2 support both cassette tape and 5.25 inch, double-density disc storage.

Table 1.2 480Z firmware versions and start-up messages

Version	Revision	Start-up message
ROS 1.0	-	RML 40-Character LINK 480Z V1.0 RML 80-Character LINK 480Z V1.0
ROS 1.1	A B	RML 40-Character LINK 480Z V1.1 A or B RML 80-Character LINK 480Z V1.1 A or B
ROS 1.2	A B C D	RML 80-Character LINK 480Z V1.2 A, B, C, or D
ROS 2.2	B	RML 80-Character LINK 480Z V2.2 B

Firmware Commands

Details of the keyboard-entered firmware commands are given in your Users Guide. Table 1.3 summarizes these commands and their functions; in general these have not changed with changes in firmware version. However, some commands are not present in all versions (see implementation in table 1.3).

Table 1.3 Firmware commands

Command	Function	Implementation
B	Boot CP/M	COS disc versions, ROS 1.2 and 2.2
X	Boot CP/M from another drive	COS disc versions, ROS 1.2 and 2.2
N	Boot network	ROS only
T	Enter terminal mode	ROS only
L	Load program from cassette	All versions except COS 4.0 and 4.2
D	Dump memory to cassette	All versions except COS 4.0 and 4.2
C	Continue program at restart address	All versions except COS 4.0 and 4.2
J	Jump to address and start program	All versions
O	Select printer option (see below)	All versions
M	Enable HRG board as memory	COS 3.4 and 4.0 only
W	Select 40 or 80-character mode	All 80-character machines
R	Enter BASIC in ROM	ROS 1.1, 1.2, and 2.2
CTRL/SHIFT/8	Interrupt program, return to most recent operating system	ROS only
CTRL/SHIFT/9	Interrupt program and enter Front Panel	ROS only
CTRL/F	Enter Front Panel	All versions
CTRL/T	Enter typewriter mode	All versions

The O command selects both printer option and cassette speed in all versions except COS 3.4/M, 3.4/F, 4.0 and 4.2 (these versions do not support cassette features).

Screen And Cursor Control

Chapter 2 gives information about some screen control and cursor control facilities that are of importance later in the manual. Definitions are given of the types of characters used on 380Z and 480Z machines.

THE EMT MECHANISM

It is very useful to be able to access the routines of the firmware, particularly for common actions like those mentioned above. This would be very easy if the firmware did not change from version to version.

Except where there are major hardware changes, the firmware routines do not change much from version to version. But the position in memory of each routine does change. Consequently, a standard way of accessing the firmware is needed, and this must work in all versions despite differences.

This is done by a mechanism that resembles the operation of a storekeeper. Rather than going into a warehouse to look for an item, you must consult a storekeeper with your request. He knows where the item is kept, and he finds it for you. Each firmware version has its own "storekeeper".

The mechanism uses a single entry point to the routines; this is called the EMT (EMulator Trap) mechanism. Access to routines is made using a two-byte call. The first byte contains the Z80 instruction, RST 30H, and the second byte contains the code number of the particular EMT routine that you want to use.

The Research Machines ZASM Assembler interprets the EMT mnemonic. The instruction:

```
emt   n
```

(where n is the number of the required instruction)

is equivalent to:

```
rst   30h
defb  n
```

in Zilog source code, or:

```
rst   6
db    n
```

in Intel code.

Full details of the EMT-calling mechanism are given in chapter 12, but here is a simple example of how two EMT instructions (KBDWF and OUTC) are used

for keyboard input and screen output:

```
        org     0100h

        kbdwf   equ 22h             ;EMT
        outc    equ 01h             ;values.

start:
        emt     kbdwf               ;Use EMT mechanism to get character
                                    ;from keyboard.
        emt     outc                ;Use EMT mechanism to send entered
                                    ;character to screen.
        jp      start               ;Repeat.

end
```

Chapters 3 to 9 give full details of the range of facilities available using EMT instructions,and these include:

- Screen handling (the OUTC family) - chapter 3
- Screen handling (other EMT instructions) - chapter 4
- Keyboard handling - chapter 5
- Printer and interface handling - chapter 6
- Cassette handling - chapter 7
- Disc handling - chapter 8
- Miscellaneous EMT instructions - chapter 9

Appendix A is a quick reference quide to all the EMT instructions.

Transferable Software

Research Machines microcomputers have evolved over the years with the addition of new facilities. As these are added, they require firmware support and, regrettably, incompatibility between systems occurs. Every effort is made to reduce such occurrences, but there are times when constraints or unforseen circumstances make them inevitable.

Such incompatibility is of little concern when software is written either in a high-level language, or for a specific system in a low-level language. However, problems occur with low-level software that is intended for use on several systems, and with high-level programs that access low-level features directly (for example when using escape sequences).

To minimise these problems of program transfer, Research Machines recommend that the firmware facilities should be accessed as follows:

- By CP/M mechanisms, where possible (see the CP/M Programmer's Guide,

PN 12084). This rules out cassette tape systems.

- By the EMT mechanism using common routines, where possible.

Details of the first implementation of EMT instructions and variations in each firmware version are given at the beginning of each of the EMT chapters (chapter 3 to chapter 9). These are presented in tabular form so that you can see at a glance which are the common EMT instructions.

Guidelines are also given, in the introduction to each chapter, about the best instructions to use for transferable software. The main areas of incompatibility are in screen handling and cassette support.

OTHER FIRMWARE FACILIITES

This section contains information about other facilities provided by the firmware, and aspects of memory layout that you will need to know about when writing low-level language programs. Full details are given in later chapters.

Debugging Facility

The firmware provides a very useful program-debugging feature. This is known as the Front Panel, and it allows you to modify the Z80 registers and the contents of memory. There are several commands that allow you to test programs for bugs. Full details are given in chapter 10.

Direct Access to Screen Memory

The normal way to send characters to the screen in your low-level programs is to use EMT instructions. However, it is possible to write to the screen by directly accessing the screen memory.

Direct access should not be used unless it is essential. For example, it can be used to switch rapidly between several displays held in RAM. Programs using this technique will not be transferable.

Chapter 11 gives details of direct access to screen memory.

Memory Layout

If you write a program that uses a lot of memory space or specialized areas of memory, you will need to know about memory layout and the areas of memory that are available. This information is provided in chapter 12.

Transfer Vectors and Device Handlers

Some EMT instructions call subroutines using "transfer vectors". These are locations in memory that contain the addresses of the subroutines.

By changing the address in the transfer vector, or by using the TRAPX vector, you can cause an EMT to call a routine of your own, a device handler. For example, you could write your own routine to control output to a printer.

Chapter 13 shows how the EMT mechanism works and how you add device handlers.

Position Independent Code

The firmware provides a relative-call instruction, CALR, that is useful when writing position independent code (PIC). Chapter 13 introduces PIC and CALR.

CONVENTIONS USED IN THIS MANUAL

This section outlines conventions that are used throughout the manual.

Bits

Several of the topics discussed in this manual refer to the bit pattern within a byte of information. Figure 1.2 below shows the standard numbering convention.

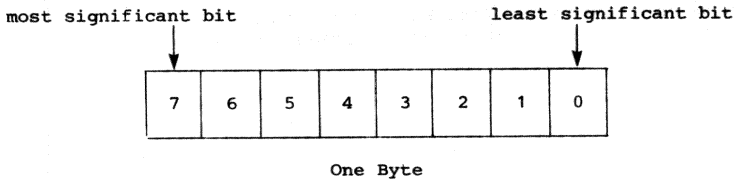


Figure 1.2 Bit numbering convention

Note that the lefthand end of the byte is most significant.

Keyboard Entry

Any specified character or characters to be keyed in at the keyboard are contained between "<" and ">". Three examples are given below:

<B> <CTRL/F> <RETURN>

Implementation Tables With Version Differences

At or near the start of chapters 3 to 10 there are summary sections. These include tables showing the firmware version in which each EMT instruction was first introduced; the sections also include any changes to the instruction in later versions that may influence whether software is transferable. The following conventions have been used in producing these tables:

- COS 3.4 is the reference version against which other implementations are compared.
- A black circle is used to signify the first implementation of an EMT instruction, except if the first implementation was in COS 3.0 or earlier. Because COS 3.4 is the reference version, the black circle is placed in the COS 3.4 column, and a "totally compatible" symbol (*) is placed in the COS 3.0 column.
- The following symbols are used to indicate version changes in terms of the transferability of programs:

*	Totally compatible. No (or minor) differences
+	Almost the same. Some differences but generally compatible.
-	Major differences. Probably incompatible.
X	Not implemented.

For example, the table below gives information about implementation and differences in cassette-handling EMT instructions. There is a row for each EMT instruction and a column for each firmware version.

Name	COS 3.0	COS 3.4/C	COS 3.4/M+F	COS 4.0	COS 4.2	ROS 1.0	ROS 1.1	ROS 1.2	ROS 2.2
PUTBYT	*	●	*	-	-	-	-	-	-
GETBYT	*	●	*	-	-	-	-	-	-
GETSYN	*	●	*	X	X	+	+	+	+
SETCAS	*	●	-	X	X	*	*	*	*
CASCTL	X	X	X	X	X	●	*	*	*

In this example, although four of the instructions were present in COS 3.0, the black circles are not placed in the COS 3.0 column; they are put in the COS 3.4 column, and stars are placed in the COS 3.0 column.

However, where instructions are not present, an X is used. For example, CASCTL was first implemented in ROS 1.0, and the previous versions all have an X in their column.

Reading along the GETSYN row, it is evident that this instruction is not implemented in COS 4.0 or 4.2. It is implemented in all versions of ROS, but with some differences from COS 3.4.

CHAPTER 2

SCREEN AND CURSOR CONTROL

This chapter outlines details of screen-handling, cursor-control and character definition on the 380Z and 480Z machines.

The first section gives screen-handling information, and the second section covers cursor control. Finally, there is a section about character definitions.

SCREEN HANDLING

With both the 380Z and 480Z machines, information is displayed on the monitor using 24 lines. There can be 40 characters per line or 80 characters per line depending on the firmware version that you have:

- Early 380Z versions (COS 3.0 and 3.4) have only 40-character mode. In COS 4.0 and 4.2, both 40 and 80-character modes are standard.
- When the 480Z was introduced, ROS 1.0 and 1.1 supported either only 40-character mode, or 40 and 80-character mode. In ROS 1.2 and 2.2, 40 and 80-character operation is standard.

Autopaging

If more than 23 lines of information are sent to the screen (for example, when listing a program), the contents of the screen scroll upwards and stop after 23 lines have been displayed. The cursor blinks at the bottom left of the screen, and the next screenful is not displayed until a key on the keyboard is pressed.

This is known as autopaging. If you want continuous screen display, autopaging can be turned off by entering <CTRL/A>. You can return to autopaging by pressing <CTRL/A> again (toggle switch). Autopaging can also be turned on or off using the control characters, CTRL/S (on) and CTRL/Q (off) - see chapter 3.

In COS 4.0 and 4.2, and all versions of ROS, there is a disable flag; when set it disables autopaging. The state of the disable flag is controlled by an escape sequence (defined in chapter 3).

NOTE: If the cursor is not in the bottom lefthand corner of the screen before you start full screen scrolling with autopaging, there will be more than 23 lines displayed before the first interruption. For example, if the cursor was positioned at the top lefthand corner, 46 lines will be displayed before the scrolling stops.

The 23 lines are counted from the last keystroke entered, not from

the last scroll position.

Smooth Scrolling

Smooth scrolling, a slower (but smooth) scrolling speed than normal scrolling, is available only in COS 4.0. The screen contents will scroll smoothly upwards only.

Windows

In all firmware versions (except COS 3.0 and 3.4) it is possible to define a rectangular area of the screen within which cursor movement and scrolling are limited. Chapter 3 gives details of how to do this with the WINDOW EMT or with an escape sequence.

CURSOR CONTROL

The rectangular cursor is normally situated at the point on the screen where the next character sent to the screen will appear. If the cursor is at the end of a line (column 39 or 79), it moves to the start of the next line (column 0). Special note is taken at this point and, if the following character is <RETURN>, it is ignored.

If the cursor is at the righthand end of the bottom line, the screen contents are scrolled upwards one line; the cursor moves to the start of the new bottom line.

Some of the facilities described in this manual allow cursor positioning at any point of the screen. For this purpose, the origin of the x and y axes is at the top lefthand corner of the screen, as shown in Figure 2.1.

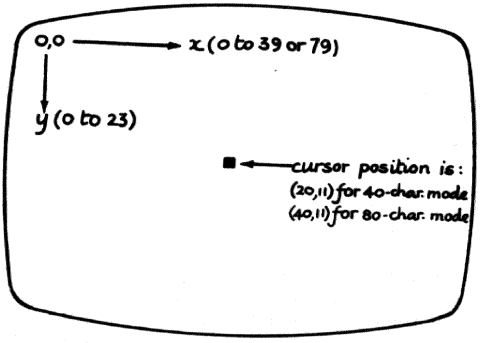


Figure 2.1 Screen cursor positioning

CHARACTER DEFINITIONS

ASCII - Coded Characters

Table 2.1 gives the seven bit ASCII-codes (0 to 127) used in Research Machines computers. Note the differences between COS 3.0 and 3.4, shown in table 2.1(a), and all other firmware versions, shown in table 2.1(b). These are at hexadecimal values 23, 5B, 5C, 5D, 5F, 60, 7B, 7D, and 78.

Table 2.1 ASCII codes used in Research Machines computers

(a) COS 3.0 and 3.4

Column → / Row ↓	0	1	2	3	4	5	6	7
0	NUL	DLE		0	@	P	—	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	£	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	←	k	¼
C	FF	FS	,	<	L	½	l	
D	CR	GS	-	=	M	→	m	¾
E	SO	RS	.	>	N	↑	n	÷
F	SI	US	/	?	O	#	o	DEL

(b) COS 4.0 and 4.2
All versions of ROS

Column → / Row ↓	0	1	2	3	4	5	6	7
0	NUL	DLE		0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	¦
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	↑	n	~
F	SI	US	/	?	O	_	o	DEL

To read the hexadecimal code for a character, take the column number then the row number from the table; for example, the character G has code 47 hex (row 4, column7) or 71 decimal.

The control codes in columns 0 and 1 (discussed in chapter 3) all have standard ASCII mnemonic names such as NUL, DLE, FF. Research Machines refers to these characters by the equivalent keyboard character, a combination of the CTRL key plus another key. If column 0 is transposed with column 4, and column 1 with column 5, the corresponding keyboard character code can be found; for example, the CTRL/F character corresponds to ACK, CTRL/Z to SUB, CTRL/L to FF, and so on.

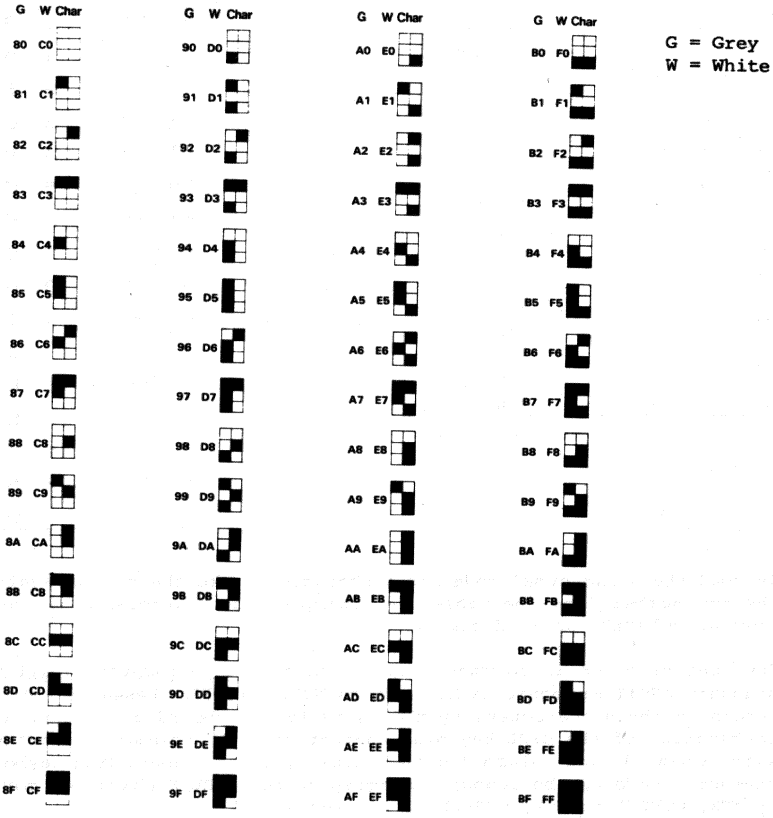
Teletext Graphics Characters

Table 2.2 shows the teletext graphics characters (128-255) that apply to the 380Z and 480Z normal character sets.

When using teletext graphics characters you should note that:

- For the 380Z,
  characters in the range 128 to 191 are displayed in grey (low intensity) on both colour and black and white monitors
- For the 480Z,
  characters 128 to 191 are displayed in grey on black on a black and white monitor, but in white on black on a colour monitor.

Table 2.2 Teletext graphics characters



In COS 4.0 and 4.2, these characters can be redefined using an escape sequence (see chapter 3).

Special Graphics Characters

The control characters of table 2.1 (0 to 31 and 127) can be plotted to give special graphics characters, as shown in table 2.3 and 2.4.

Table 2.3 Special graphics characters (COS 3.0 and 3.4)

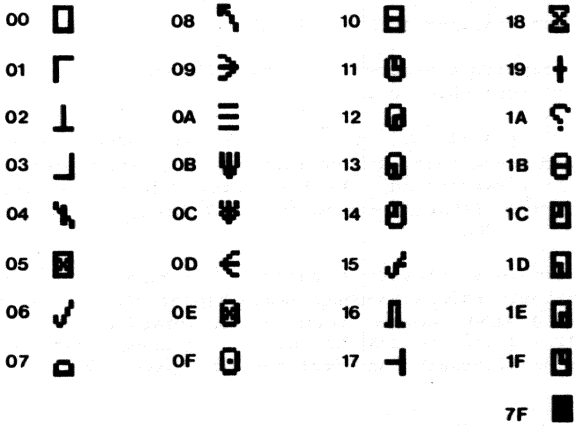
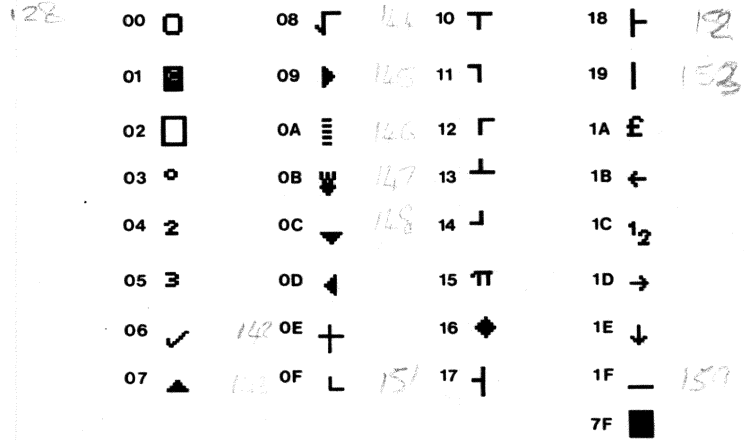


Table 2.4 Special graphics characters (other firmware versions)



These special graphics characters are specified in EMT instructions that access the screen (see chapter 4), and (in COS 4.0 and 4.2, and in all ROS versions) by using an escape sequence (see chapter 3).

Special Features

There are some special features which modify characters. These features are not available in all firmware versions:

- Redefinition of teletext graphics characters (COS 4.0 and 4.2 only).
- Character attributes (COS 4.0 and 4.2 only).
- Alternate characters (ROS 2.2 only).

The teletext graphics characters (128 to 255) can be redefined using an escape sequence (see chapter 3).

Character attributes can be set with an escape sequence (see chapter 3) to give underlined characters, dim characters, or reverse-video characters (black on a white background). In 80-character mode, the automatic dimming attribute switches off or on the automatic dimming of teletext graphics characters 128 to 191.

In ROS 2.2 there is an alternate character set that can be selected by the character attribute escape sequence (see chapter 3). The alternate set has reverse-video, dimmed, special graphics characters (0 to 31) and ASCII characters (32 - 126). In COS 4.0 and 4.2, an escape sequence attribute is provided, but no alternate character set is implemented.

CHAPTER 3

SCREEN HANDLING - THE OUTC FAMILY

This chapter gives details of the OUTC family of EMT instructions. These instructions are:

- OUTC, OUTCNV, MSG, and OUTNC
- CURPOS, GRAFIX, SCROLL, WIDTH, and WINDOW.

The first group are instructions that send characters to the screen. The second group can be considered as maintenance EMTs that are used to find the cursor address, to modify scrolling behaviour and screen mode, and to define window area.

The instructions detailed in this chapter give you a large number of screen-handling facilities with compatibility between firmware versions. You are strongly advised to use these instructions in preference to any of the other screen-handling methods mentioned in this manual.

The first section of the chapter summarizes the OUTC family of instructions and version differences; the second section gives detailed definitions. Finally, there are sections on control characters and escape sequences.

SUMMARY OF INSTRUCTIONS AND VERSION DIFFERENCES

Table 3.1 summarises the OUTC family of EMT instructions.

Table 3.1 The OUTC family of EMT instructions

Mnemonic	Code (hex)	Function
OUTC	1 (01)	Output the byte in register A to the screen
OUTCNV	22 (16)	As above but not using a transfer vector
MSG	23 (17)	Output a message at (HL) to the screen
OUTNC	42 (2A)	Output a byte without clearing the page key
CURPOS	59 (3B)	Return the address of the cursor position (x,y)
GRAFIX	13 (0D)	Clear the top 20 lines giving a 4-line scroll
SCROLL	14 (0E)	Full screen scroll
WIDTH	52 (34)	Change screen mode or return present state
WINDOW	58 (3A)	Define window area of the cursor

Table 3.2 gives implementation details and version differences of these EMT instructions.

Table 3.2 Implementation details and version differences

Name	Code	COS 3.0	COS 3.4	COS 4.0	COS 4.2	ROS 1.0	ROS 1.1	ROS 1.2	ROS 2.2
OUTC	1	+	●	+	+	+	+	+	+
OUTCNV	22	+	●	+	+	+	+	+	+
MSG	23	*	●	*	*	*	*	*	*
OUTNC	42	*	●	*	*	-	-	-	-
CURPOS	59	X	X	●	*	*	*	*	*
GRAFIX	13	+	●	+	+	+	+	+	+
SCROLL	14	*	●	+	+	+	+	+	+
WIDTH	52	X	X	●	*	+	+	+	+
WINDOW	58	X	X	●	*	+	+	+	+

● Implemented
* Fully compatible
\+ Almost the same
\- Major changes
X Not implemented

Note that, although OUTC is implemented on all versions, there have been changes in the control characters that it interprets. Also, escape sequences were not interpreted until COS 4.0, and there are differences between COS and ROS escape sequences. See the relevant sections in this chapter for details.

DEFINITIONS

OUTC

Output the byte in register A to the screen

Decimal code (hex) : 1 (1)

Implementation : COS 3.0 and 3.4

Registers affected : None

OUTC outputs the character given in register A to the screen at the current cursor position. The cursor moves one place to the right.

In all firmware versions, OUTC interprets a byte as ASCII characters (32 to 126, table 2.1 in chapter 2), and teletext graphics characters (128 to 255, table 2.2 in chapter 2).

OUTC is unique in interpreting control characters (0 to 31 and 127, table 2.1 in chapter 2); these provide screen and cursor-handling facilities. There have been changes in control characters interpretation with firmware differences. Consult the section on control characters (later in this chapter) for details.

In COS 3.0 the cursor is bound to the bottom line.

This instruction is called using a transfer vector (see chapter 13 for its offset address).

OUTC modifications

Implementation : COS 4.0 and 4.2.
All versions of ROS

The instruction automatically takes account of whether you are in 40 or 80-character mode, and whether to do ordinary scrolling or (COS 4.0 only) smooth scrolling.

Escape sequence commands are interpreted by OUTC when ESC (27) is received. The section on escape sequences (later in this chapter) gives details.

OUTCNV

As above but not using a transfer vector

Decimal code (hex) : 22 (16)

Implementation : All versions of COS and ROS

Registers affected : None

This instruction works in the same way as above, except that the call does not use a transfer vector. In general, the use of this EMT is not recommended.

MSG

Output a message at (HL) to the screen

Decimal code (hex) : 23 (17)

Implementation : All versions of COS and ROS

Registers affected : None

MSG is used to output a string of characters to the screen starting at the current cursor position. Register HL must be set to point to the start of the string; the end must be

represented by a negative byte (with the most significant bit set) which is not displayed.

OUTNC

Output a byte without losing autopaging characters

Decimal code (hex) : 42 (2A)

Implementation : All versions of COS

Registers affected : None

OUTNC operates in the same way as OUTC except when autopaging is active (see chapter 2).

When a key is pressed to obtain more text (in response to the flashing cursor), the character entered is not cleared from the keyboard latch (except for <CTRL/A>).

OUTNC modification

Implementation : All versions of ROS

OUTNC is identical to OUTC

CURPOS

Returns the address of the cursor position (x, y)

Decimal code (hex) : 59 (3B)

Implementation : COS 4.0 and 4.2.
All versions of ROS

Registers affected : HL

CURPOS returns the address of the cursor position with respect to the top lefthand corner of the screen (see chapter 2). Register H contains the y value, and register L contains the x value.

GRAFIX

Clear the top 20 lines giving a 4-line scroll

Decimal code (hex) : 13 (0D)

Implementation : All versions of COS and ROS

Registers affected : None

GRAFIX clears the top 20 lines of the screen and confines scrolling to the bottom 4 lines. In COS 4.0 and 4.2, the cursor is moved to the bottom left of the screen.

There are some version differences:

- In COS 4.0 and COS 4.2, and all versions of ROS, this instruction works in 40-character or 80-character mode
- Autopaging is automatically disabled in COS 4.0, COS 4.2, and all versions of ROS, but not in COS 3.0 and COS 3.4. Consequently, autopaging should be separately disabled in the latter versions.

SCROLL

Full screen scroll

Decimal code (hex) : 14 (0E)

Implementation : All versions of COS and ROS

Registers affected : None

SCROLL restores full screen scrolling. In COS 3.0 and 3.4, and all versions of ROS, the cursor is unaffected. In COS 4.0 and 4.2, the cursor is placed at the bottom left of the screen.

This instruction works in 40-character and 80-character mode.

If autopaging was automatically disabled by the GRAFIX EMT, it is automatically re-enabled.

WIDTH

Change screen mode or return present state

Decimal code (hex) : 52 (34)

Implementation : COS 4.0 and 4.2

Register affected : A and the flags

WIDTH can be used to change the screen character mode (in COS 4.0 and 4.2, the cursor in moved to the bottom left of the screen), or to return the current mode.

- To change modes, register A must be set before entry to WIDTH. If the register holds 0, the 40-character mode will be selected; if 1, the 80-character mode will be selected.

If the change is from 40 to 80, the 40-character data is displayed on the lefthand side of the 80-character screen, and the righthand side is blank. If the change is from 80 to 40, each 80-character line is truncated to become a new-line.

- To return the current mode, register A must contain -1 on entry. On return, it will hold 0 if the screen is in 40-character mode, and 1 if the screen is in 80-character mode.

WIDTH modifications

Implementation : All versions of ROS

Registers affected : A, BC, DE and the flags.

WIDTH operates as above, but there is an additional facility to return the limits of the current scrolling window. If register A contains -2 on entry, then, on return, the x, y limits of the window are held in the B, C, D and E registers:

- Lower y value (top line) in register B
- Upper y value in register C
- Lower x value (leftmost) in register D
- Upper x value in register E.

WINDOW

Define window area of the cursor

Decimal code (hex) : 58 (3A)

Implementation : COS 4.0 and 4.2.
All versions of ROS

Registers affected : None

WINDOW defines the area of the screen to which cursor movement and scrolling are limited.

On entry, registers B, C, D and E must hold the x, y limits of the window:

- lower y value (top line) in register B
- upper y value in register C
- lower x value (leftmost) in register D

- upper x value in register E.

The following conditions must apply:

```
0<=B<=C<=23

0<=D<=E<=39 (40-character mode) or
         79 (80-character mode)
```

If any value does not conform, the EMT has no effect.

In COS 4.0 and 4.2, after the new window has been defined, the cursor is placed at the bottom left corner of the window. This occurs no matter where the cursor was before the call.

In ROS, the cursor remains in the same position unless it is outside the new window; in this case, the cursor goes to the top left corner of the window.

WINDOW works in both 40 and 80-character mode.

CONTROL CHARACTERS

The control characters that are interpreted by the OUTC family are summarized in Table 3.3. This also shows firmware implementation details. Note that very few were implemented in COS 3.0. Most of the characters are used for cursor and screen control.

Table 3.3 Summary of the control characters, and firmware implementation

Control Character	ASCII Name and Code	Function	COS 3.0	COS 3.4	COS 4.0	COS 4.2	ROS 1.0	ROS 1.1	ROS 1.2	ROS 2.2
CTRL/@	NUL (0)	Reserved 'NULL'								
CTRL/A	SOH (1)									
CTRL/B	STX (2)									
CTRL/C	ETX (3)									
CTRL/D	EOT (4)	Resume output	X	●	*	*	*	*	*	*
CTRL/E	ENQ (5)									
CTRL/F	ACK (6)									
CTRL/G	BEL (7)	Beep	x	x	●	*	*	*	*	*
CTRL/H	BS (8)	Cursor left	x	●	*	*	*	*	*	*
CTRL/I	HT (9)	Tab	*	●	*	*	*	*	*	*
CTRL/J	LF (10)	Cursor down	*	●	+	+	+	+	+	+
CTRL/K	VT (11)	Cursor up	X	●	+	+	+	+	+	+
CTRL/L	FF (12)	Clear screen (BL)	*	●	*	*	*	*	*	*
CTRL/M	CR (13)	CR + LF	*	●	*	*	*	*	*	*
CTRL/N	SO (14)	CR	*	●	*	*	*	*	*	*
CTRL/O	SI (15)	Suppress output	X	●	*	*	*	*	*	*
CTRL/P	DLE (16)									
CTRL/Q	DC1 (17)	Stop autopaging	*	●	*	*	*	*	*	*
CTRL/R	DC2 (18)	Reverse video on	X	X	●	*	X	X	X	X
CTRL/S	DC3 (19)	Start autopaging	*	●	*	*	*	*	*	*
CTRL/T	DC4 (20)	Reverse video off	X	X	●	*	X	X	X	X
CTRL/U	NAK (21)	Blinking cursor on	X	●	*	*	*	*	*	*
CTRL/V	SYN (22)	Cursor addressing	X	●	+	+	+	+	+	+
CTRL/W	ETB (23)	Blinking cursor off	X	●	*	*	*	*	*	*
CTRL/X	CAN (24)	Cursor right	X	●	*	*	*	*	*	*
CTRL/Y	EM (25)	Delete to end of line	X	●	*	*	*	*	*	*
CTRL/Z	SUB (26)									
CTRL/[	ESC (27)	Start escape sequence	X	X	●	*	*	*	*	*
CTRL/	FS (28)									
CTRL/]	GS (29)	Cursor home	X	●	*	*	*	*	*	*
CTRL/	RS (30)	Clear to end of screen	X	●	*	*	*	*	*	*
CTRL/_	US (31)	Cursor home + clear sc.	X	●	*	*	*	*	*	*
	DEL (127)	Backspace + delete	*	●	*	*	*	*	*	*

● Implemented
* Fully compatible
\+ Almost the same
X Not implemented

Definitions

CTRL/D 04 hex (EOT)

Resume output

This character causes output to resume, clearing the effect of CTRL/O which suppresses output.

CTRL/G 07 hex (BEL)

Sound the Beeper

In all 380Z systems the hardware does not include a speaker. One can be attached between pin 6 (beeper) and pin 7 (ground) of the 7-way DIN socket marked 'Cassette Recorder' on the rear panel.

CTRL/H 08 hex (BS)

Cursor left

This character moves the cursor left one position. If the cursor was at the beginning of a line, it moves to the end of the previous line. If the cursor was at the beginning of the top line, it moves to the end of that line, and the screen contents are scrolled down one line (not in COS 3.0 or 3.4).

CTRL/I 09 hex (HT)

Horizontal tab

The cursor moves right to the next tab position.

CTRL/J 0A hex (LF)

Cursor down

The cursor moves vertically down one line. If this moves the cursor off the screen, the contents of the screen will be scrolled up one line (in COS 3.4, this also results in cursor movement to the beginning of this new bottom line).

This character is ignored if it immediately follows a CTRL/M character.

CTRL/K 0B hex (VT)

Cursor up

The cursor moves up one line. If the cursor was on the top line, the contents of the screen will scroll down one line (except in COS 3.4).

CTRL/L (FF) **0C hex**

Clear screen and move the cursor to bottom left

The screen contents are cleared and the cursor is positioned at the bottom left of the screen.

CTRL/M (CR) **0D hex**

Carriage return and line feed

The cursor moves to the beginning of the next line. If the cursor was on the bottom line, the screen contents will scroll upwards one line.

This character is ignored if it immediately follows a "forced" CR/LF (see page 2.2).

CTRL/N (SO) **0E hex**

Carriage return

The cursor moves to the start of the current line.

CTRL/O (SI) **0F hex**

Suppress output

Any character (including escape sequences) sent to the screen by the OUTC EMT will not appear. Output is resumed with CTRL/D.

CTRL/Q (DC1) **11 hex**

Stop autopaging

This character stops autopaging.

CTRL/R (DC2) **12 hex**

Switch reverse-video on

In COS 4.0 and 4.2, reverse-video (see escape sequences) is set on.

CTRL/S (DC3) **13 hex**

Start autopaging

This character starts autopaging.

CTRL/T (DC4) **14 hex**

Switch reverse-video off

In COS 4.0 and 4.2, reverse-video is set off.

CTRL/U (NAK) **15 hex**

Blink on

This character starts the cursor blinking when keyboard input from KBDW or KBDWF (see chapter 5) is expected. The solid white cursor blinks on and off over any character that is underneath

it. When keyboard input is not expected, no cursor is visible.

CTRL/V (SYN) **16 hex**

Cursor addressing

This character initiates cursor addressing. It must be followed by two characters which define the vertical (y) and horizontal (x) cursor coordinates, respectively. Each of these characters consists of the cursor y or x position with decimal 32 added to it.

For example, the top left of the screen is defined by:

(0 + 32) (0 + 32)

and the bottom right of the screen is defined by:

(23 + 32) (39 + 32) in 40-character mode

or

(23 + 32) (79 + 32) in 80-character mode.

The coordinates can be defined within a window. In COS 4.0 and 4.2, the origin is the top lefthand corner of the window. But, in all versions of ROS (as in COS 3.4), the origin is the top lefthand corner of the screen.

Once CTRL/V has been received, the cursor addressing sequence must be completed.

CTRL/W (ETB) **17 hex**

Blink off

This character disables cursor blinking.

CTRL/X (CAN) **18 hex**

Cursor right

The cursor moves one space to the right. If the cursor was at the end of a line it will move to the start of the next line; the screen contents will be scrolled upwards, if necessary.

CTRL/Y (EM) **19 hex**

Delete to end of line

The screen contents are cleared to the end of the current line, and the cursor position is left unchanged.

CTRL/[(ESC) **1B hex**

Start escape sequence

Receipt of this character (in COS 4.0, 4.2 and all versions of ROS) initiates a command sequence determined by the bytes immediately following CTRL/[. Refer to the next section for descriptions of the escape sequences.

CTRL/] (GS) **1D hex**

Cursor home

The cursor is moved to the top left corner of the screen, the home position.

CTRL/↑ (RS) **1E hex**

Clear to end of screen

The screen contents are cleared below and to the right of the cursor. The cursor position is left unchanged.

CTRL/_ (US) **1F hex**

Cursor home and clear screen

The screen contents are cleared, leaving the cursor at the top left corner of the screen.

DELT (DEL) **7F hex**

Backspace and delete

The cursor moves one position to the left and deletes the character at that position. If the cursor is already at the left hand edge of the screen, the cursor does not move, but the character at the cursor position is deleted.

ESCAPE SEQUENCES

When the control character ESC (27) is received by one of the OUTC family of EMT instructions (in COS 4.0 and 4.2, and in all versions of ROS), a command sequence is initiated. The series of characters which follow ESC are not sent to the screen; instead other actions take place, as can be seen from Table 3.4 which summarises the escape sequence commands and implementation differences.

Table 3.4 Summary of escape sequence commands

Sequence Introducer			Definition of Command and Firmware Implementation	
ASCII Character	Hex	Decimal	COS 4.0 and 4.2	All ROS versions
!	21	33	Send a special graphics character to the screen	Send a special graphics character to the screen
%	25	37	Not implemented	Define new uses of the function keypad keys
<	3C	60	Re-define teletext graphics characters	Not implemented
=	3D	61	Screen control (includes smooth scrolling in COS 4.0 only)	Screen control commands
>	3E	62	Initialization of various screen parameters (see p.3.19)	Restore original use of of function keypad keys
?	3F	63	Define scrolling window dimensions	Define scrolling window dimensions
@	40	64	Alter sound of beeper	Alter sound of beeper

Note that the sequence introducer (SI) is referred to in Table 3.4. This is the first of the series of characters which follow ESC. Its value determines how many subsequent characters can be part of an escape sequence. Subsequent characters needed for each SI are listed in the definitions which follow; you will encounter:

- a switch (SW) with possible values of "0" (30 HEX) to switch a feature off or "1" (31 hex) to switch it on

- a control parameter (CP) which may take any value given in the definitions.

Definitions

SI:33
(21 hex)

Send a special graphics character to the screen

ASCII character : !

Implementation : COS 4.0 and 4.2
All versions of ROS

The sequence after the SI is a single byte to be output directly to the screen; the characters 0 to 31 and 127 are not interpreted as control characters but as the special graphics characters shown in tables 2.3 and 2.4.

For example, the following sequence:

27	33	21
ESC	SI	character

will send a pi symbol to the screen. (21, within this escape sequence, represents the pi symbol)

SI:37
(25 hex)

New uses of the function keypad keys

ASCII character : %

Implementation : All versions of ROS

The sequence after the SI is a CP, followed by a byte (of value n), then n further bytes. The sequence is used to redefine the character string that is generated when a function or arrow key is pressed. The new character string will be the last n bytes of the escape sequence.

CP = 65 (A)	Redefine the up arrow key
CP = 66 (B)	Redefine the right arrow key
CP = 67 (C)	Redefine the down arrow key
CP = 68 (D)	Redefine the left arrow key
CP = 69 (E)	Redefine the F1 function key
CP = 70 (F)	Redefine the F2 function key
CP = 71 (G)	Redefine the F3 function key
CP = 72 (H)	Redefine the F4 function key
CP = 73 (I)	Redefine the SHIFT/up arrow key
CP = 74 (J)	Redefine the SHIFT/right arrow key
CP = 75 (K)	Redefine the SHIFT/down arrow key

CP = 76 (L) Redefine the SHIFT/left arrow key
CP = 77 (M) Redefine the SHIFT/F1 function key
CP = 78 (N) Redefine the SHIFT/F2 function key
CP = 79 (O) Redefine the SHIFT/F3 function key
CP = 80 (P) Redefine the SHIFT/F4 function key

The character string can be of any length from 0 to 127 bytes, but the total number of characters for all keys is also limited to 127. If the string is too long, the escape sequence will have no effect.

This is very useful in high-level languages. For example, in BASIC:

```
PUT 27,"%G",4,"RUN",13
```

redefines the F3 function key so that it generates the string:

```
RUN <RETURN>
```

SI:60
(3C hex)

Re-definition of teletext graphics characters

ASCII character : <

Implementation : COS 4.0 and 4.2

The characters from 128 to 255 are normally the teletext graphics characters. However, some or all of them can be redefined.

To define a character requires 12 bytes. A character is 8 dots wide by 10 deep. The first 10 bytes correspond to the 10 lines of dots. Every bit which is set will give a dot in the corresponding position. The 11th and 12th bytes are used for the underline attribute (see screen control and character attributes), and replace lines 9 and 10 when the underline attribute is on. To define a character, use the escape sequence:

27, 60, C, b1, b2, b3, b4, b5, b6, b7, b8, b9, b10, b11, b12

where C is the code of the character to be defined, and b1 to b12 are the bytes to define the character. For example, character 192 can be redefined as the Greek character 'rho' by:

27, 60, 192, 0, 0, 0, 0, 24, 36, 68, 120, 64, 64, 255, 64.

It should be noted that characters which are in range 128 to 191 automatically appear dim on the screen, by hardware in 40-character mode and by software in 80-character mode. It is possible to switch this feature on or off in 80-character mode. If automatic dimming is switched off, then characters which are normally dim will now be bright. The escape sequence which switches dimming off is:

27, 61, 48, 75

The graphics character store can be reinitialized to contain the usual teletext graphics characters by the escape sequence:

27, 62, 66

SI:61
(3D hex)

Screen control and character attributes

ASCII character : =

Implementation : COS 4.0 and 4.2
All versions of ROS

The sequence after the SI is a SW followed by a CP:

27, 61, SW, CP

The CP value controls which feature will be used; each feature is switched on or off by the SW value (SW= 1(31 hex) is on; SW=0 (30 hex) is off). Table 3.5 summarizes the commands and implementation differences.

For instance, the sequence:

27, 61, 75, 49

Switches on automatic dimming of teletext graphics characters in 80-character mode (COS 4.0 and 4.2 only).

Table 3.5 Sequence introducers for screen control

CP	Function	Implementation COS 4.0	COS 4.2	ALL ROS
65 (A)	Alternate character set	●	*	X
66 (B)	Underline attribute	●	*	X
67 (C)	Dim attribute	●	*	X
68 (D)	Reverse-video attribute	●	*	X
70 (F)	Front Panel entry	●	*	*
71 (G)	Autopaging	●	*	*
72 (H)	Smooth scrolling	●	X	X
73 (I)	Text in HRG output	●	*	X
74 (J)	40 or 80 character mode	●	*	*
75 (K)	Automatic dimming	●	*	X
76 (L)	Alternate character set	X	X	●

● Implemented X Not implemented
* Fully compatible

The functions referred to in table 3.5 are defined below:

- smooth scrolling (COS 4.0 only) - a slower, but smooth, scrolling speed. Switched on or off by typing <CTRL/@>, or by the escape sequence.

- underline, dim, reverse video, automatic dimming, (COS 4.0 and 4.2 only) - Characters sent to the screen can be underlined, dimmed, or displayed as dark on light (reverse video). Reverse video can be switched on by <CTRL/R> and off by <CTRL/T>. Automatic dimming switches off or on the automatic dimming of teletext graphics characters, 128 to 191 (in 80-character mode).

- switch on or off the text in high resolution graphics output (COS 4.0 and 4.2 only) - Only graphics is output when text is switched off.

- alternate characters (COS 4.0 and 4.2, and ROS 1.0, 1.1, and 1.2) - An escape sequence attribute is provided (CP=65 in COS, CP= 76 in ROS) but no alternate character set is implemented. If you attempt to use this attribute, undefined results will occur in COS, and the normal set will be produced in ROS.

(ROS 2.2) - There is an alternate character set that can be selected by the escape sequence attribute (SI=61, CP=76). This set has the teletext graphics characters replaced by a reverse dim version of the normal set:

Character Codes	Alternate Character Set
0-31	Special graphics characters, bright
32-127	ASCII characters, bright
128	Blank
129-159	Special graphics characters, reverse dim
160-255	ASCII characters, reverse-dim

SI:62
(3E HEX)

Initialize character attributes

ASCII character : >

Implementation : COS 4.0 and 4.2

The sequence after the SI is a CP only:

- CP=65 (A) clears the current attributes, clears smooth scrolling (COS 4.0 only), sets to 80-character mode, places the cursor at the bottom left of the screen and initializes the teletext characters to their usual value
- CP=66 (B) initializes the teletext characters to their usual values
- CP=67 (C) clears the current attributes

SI:62
(3E HEX)

Initialize and restore function keypad keys

ASCII character : >

Implementation : All versions of ROS

The sequence after the SI is a CP only:

- CP=68 (D) restores all function and arrow keys to their original values

SI:63
(3F hex)

Define or redefine scrolling window dimensions

ASCII character : ?

Implementation : COS 4.0 and 4.2.
All versions of ROS.

The sequence after the SI is 4 bytes followed by a CP:

- CP=65 (A).

  The 4 bytes define the 4 corner positions of a scrolling window:

  x(lower),x(upper),y(lower),y(upper)

  The sequence is:

  27, 63, xl, xu, yl, yu, CP
  (where l=lower, u=upper)

  The values must be such that:

  0 <=xl <=xu <=39 (40-character mode)
  or
  0 <=xl <=xu <=79 (80-character mode)
  and
  0 <=yl <=yu <=23.

- CP = 66 (B)

  The 4 bytes define a rectangular area of the screen to be cleared, with order and value limits as above.

See also the EMT WINDOW for further information about windows.

SI:64
(40 hex)

Alter sound of beeper

ASCII character : @

Implementation : COS 4.0 and 4.2.
All versions of ROS

The sequence after the SI is 2 bytes followed by a CP, which is 41 hex (A):

27, 64, F, d, 65

The 2 bytes define the frequency (F) and duration (d) of the sound produced by the internal loudspeaker (480Z) or the attached loudspeaker (380Z). The desired sound can be calculated from the formula:

$$\text{Frequency (kHz)} = \frac{1000}{54 + 32*F}$$

CHAPTER 4

SCREEN HANDLING - OTHER EMT INSTRUCTIONS

This chapter gives details of EMT instructions that access the screen memory and are faster than the OUTC family of EMT instructions. However, if you use them, you may have problems in firmware compatibility; we recommend that you use OUTC instructions rather than these EMTs.

A major compatibility problem is caused by differences in hardware between the 380Z and 480Z machines:

- In the 380Z, screen display is held in RAM; this is accessible to both the monitor circuitry and the CPU, though not simultaneously. The CPU can access the screen memory at any time, but if this occurs while information is being written to the screen the picture will be disturbed. To prevent this happening, the CPU only accesses the screen memory when nothing is being written to the screen (that is, during the line and frame blanking periods).

- The 480Z screen memory is mapped as a block of 24 Z80 input/output ports each corresponding to a line on your screen, and simultaneous access by both monitor circuitry and the CPU is possible.

These hardware differences (see the Information File manuals for details) mean that the EMT instructions are in two main groups: one for the 480Z, and the other for the 380Z.

In the first section of this chapter, the EMT instructions are summarized with details of implementation and version differences. The second section gives definitions.

SUMMARY OF INSTRUCTIONS AND VERSION DIFFERENCES

Table 4.1 summarizes the memory-access, screen-handling EMT instructions.

Table 4.1 Screen-handling (memory access) EMT instructions

Mnemonic	Code (hex)	Function
VTOUT	55 (37)	Output character to screen
VTIN	56 (38)	Read character from screen
VTCLR	57 (39)	Clear specified area of the screen
VTLINE	60 (3A)	Output line or part of line
OPNWT	11 (0B)	Open screen for memory access
CLOSE	12 (0C)	Close screen
CLEAR	15 (0F)	Clear selected screen band
OUTF	43 (2B)	Output to screen at (HL) from A reg.
INF	44 (2C)	Input from screen at (HL) to A reg.
CHGEN	53 (35)	Generate new character pattern
CHREAD	54 (36)	Read current character pattern

Table 4.2 gives details of implementation and firmware differences.

Table 4.2 Implementation details and differences

Name	Code	COS 3.0	COS 3.4	COS 4.0	COS 4.2	ROS 1.0	ROS 1.1	ROS 1.2	ROS 2.2
VTOUT	55	X	X	●	*	-	-	-	-
VTIN	56	X	X	●	*	-	-	-	-
VTCLR	57	X	X	●	*	+	+	+	+
VTLINE	60	X	X	●	*	+	+	+	+
OPNWT	11	*	●	*	*	-	-	-	-
CLOSE	12	*	●	*	*	-	-	-	-
CLEAR	15	*	●	+	+	X	X	X	X
OUTF	43	*	●	*	*	X	X	X	X
INF	44	*	●	*	*	X	X	X	X
CHGEN	53	X	X	●	*	X	X	X	X
CHREAD	54	X	X	●	*	X	X	X	X

●	Implemented	-	Major differences
*	Fully compatible	X	Not implemented
+	Almost the same		

VTOUT, VTIN, VTCLR, and VTLINE access the screen memory. Characters can be sent to any position on the screen more directly with VTOUT and VTLINE than they can with OUTC or MSG (see Chapter 3). However, these memory-access EMT instructions do not interpret the codes 0 to 31 and 127 as control codes; instead, the special graphics characters shown in Table 2.4 are sent to the screen.

OPNWT, CLOSE, CLEAR, OUTF, and INF are used with COS firmware. They form a second group that synchronizes access of the CPU to the screen memory during line and page blanking. INF, OUTF and CLEAR were deleted from ROS when the new VTOUT, VTIN and VTCLR instructions replaced them. OPNWT and CLOSE were retained for compatibility, but their only function is to synchronize the blanking period.

The final two instructions, CHGEN and CHREAD, are implemented only in COS 4.0 and 4.2, 80-character mode.

DEFINITIONS

VTOUT — Output character to screen

Decimal code (hex) : 55(37)

Implementation : COS 4.0 and 4.2

Registers affected : None

VTOUT outputs a character to the screen at a specified address. This instruction works in both 40 and 80-character mode. Register A contains the character to be output, and register HL contains the x, y address: the y value (0 to 23) is in H; the x value (0 to 39 or 0 to 79) is in L. If the coordinates are outside these limits, the instruction has no effect.

Register E contains the character display attribute. For normal character display, E contains zero. To obtain other attributes, set the appropriate bit:

0	:	alternate ROM
1	:	underline
2	:	dim characters
3	:	reverse video
4 to 7	:	ignored

VTOUT modification

Implementation : All versions of ROS

Registers affected : None

VTOUT works in the same way as described above, except that there are no character display attributes in register E.

It works in 40-character mode or 40 and 80-character mode machines.

VTIN

Read character from screen

Decimal code (hex) : 56 (38)

Implementation : COS 4.0 and 4.2

Registers affected : A, E, and the flags

VTIN reads a character from the screen at a specified address. This instruction works in 40 or 80-character mode. Register HL contains the x, y coordinates of the address: the y value (0 to 23) is in H; the x value (0 to 39, or 0 to 79) is in L. If the coordinates are outside these limits, then registers A and E contain zero on return, otherwise register A contains the character at the address, and register E contains the character display attributes information (see the VTOUT definition).

VTIN modification

Implementation : All versions of ROS

Registers affected : A, E and the flags

VTIN works in the same way as described above, except that, on return, register E contains zero (a 480Z has no display attributes).

VTIN works in both 40 and 40/80 character mode machines.

VTCLR

Clear specified area of the screen

Decimal code (hex) : 57 (39)

Implementation : COS 4.0 and 4.2, and all versions of ROS

Registers affected : None

VTCLR clears a specified area of the screen. This instruction works in 40-character machines and in 40/80-character machines.

On entry, registers B, C, D, and E must hold the x, y limits of the area:

- lower y value (top line) in register B
- upper y value in register C
- lower x value (left most) in register D
- upper x value in register E

The following conditions must apply:

0 <=B<=C<=23

0 <=D<=E<=39 (40-character mode) or
79 (80-character mode)

If any value does not conform, the EMT has no effect.

Those positions on the screen that have been cleared by this instruction display a blank character (80 hex).

VTLINE

Output line or part of line

Decimal code (hex) : 60 (3A)

Implementation : COS 4.0 and 4.2, and all versions of ROS

Registers affected : None

VTLINE outputs a line, or part of a line, to a specified address. Register HL contains the start address on the screen (in x, y form); this does not have to be the start of a line.

Register DE contains the start address of the string to be output, and register C contains the number of characters in the string. Output will stop if the end of the screen line is reached.

OPNWT

Open screen for memory access

Decimal code (hex) : 11 (0B)

Implementation : All versions of COS

Registers affected : None

OPNWT waits for the beginning of a frame blanking period, then allows the CPU to access screen memory. The screen will go blank when screen memory is open to the CPU, so this should only be done during frame blanking.

OPNWT modification

Implementation : All versions of ROS

Registers affected : None

A call to OPNWT has no effect other than to wait for the next frame blanking signal.

CLOSE

Close screen

Decimal code (hex) : 12 (0C)

Implementation : All versions of COS

Registers affected : None

The screen memory is closed to prohibit access by the CPU, and to allow the display to be shown again.

CLOSE modification

Implementation : All versions of ROS

Registers affected : None

CLOSE is implemented, but does not do anything.

CLEAR

Clear selected screen band

Decimal code (hex) : 15 (0F)

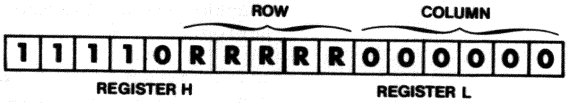
Implementation : COS 3.0 and 3.4

Registers affected : A, B, DE and HL, and the flags.

CLEAR clears a number of lines of the screen. The validity of the arguments is not checked; consequently, take care or you will get

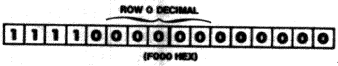
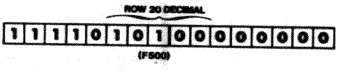
unpredictable results!

On entry, register A contains the number of lines to be cleared (1 to 24 decimal). Register pair HL contains the address of the beginning of the top line of the band to be cleared: a row number (0 to 23 decimal) and column zero (the lefthand column) must be specified in the form:



Only the five R bits will vary, with the top row (row 0 decimal) represented by 00000, the next row by 00001, the next by 00010, and so on until the bottom row (row 23 decimal) which is 10111.

Two examples of this are:

Register A	Register HL	Action
18 hex (24 decimal)		Clears the whole screen
4 hex (4 decimal)		Clears the bottom 4 lines.

On return, register A contains 0, and register HL contains the address of the position following that of the last cleared character. A blank graphics character (128) is displayed at all clear positions. The contents of registers B and DE will be destroyed.

CLEAR modification

Implementation : COS 4.0 and 4.2

Registers affected : A, B, DE, HL and the flags

In COS 4.0 CLEAR does not work, but the contents of all registers are preserved. In COS 4.2, the instruction works as before, but only in 40-character mode.

OUTF

Output to screen at (HL) from A register

Decimal code (hex) : 43 (2B)

Implementation : All versions of COS

Registers affected : None

The character in register A is written to the memory address contained in register pair HL (see chapter 11) during the line blanking period.

This instruction only works in 40-character mode. OUTF is very slow in COS 4.0 and 4.2.

INF

Input from screen at (HL) to register A

Decimal code (hex) : 44 (2C)

Implementation : All versions of COS

Registers affected : A

INF reads the memory location, the address of which is in register pair HL (see chapter 11), then places the result in register A. This is done during line blanking.

This instruction only works in 40-character mode. INF is very slow in COS 4.0 and 4.2.

CHGEN

Generate new character pattern

Decimal code (hex) : 53 (35)

Implementation : COS 4.0 and 4.2

Registers affected : None

CHGEN generates a new character dot pattern for any of the teletext graphics characters (80 to FF hex).

On entry, register A contains the character to be changed, and register B contains 0 for normal use (waiting for frame blanking) or 1 for immediate action (which may make the screen flicker).

The new dot pattern (see re-definition of graphics characters in the Escape Sequences

section of chapter 3) must be contained in 12 adjacent bytes of memory; the address of the first byte is held in register DE.

You are advised against redefining character 80 hex (128), as this is used by COS as a "blank".

CHREAD

Read current character pattern

Decimal code (hex) : 54 (36)

Implementation : COS 4.0 and 4.2

Registers affected : None

CHREAD reads the current dot pattern for a character in the range 0 to FF hex.

On entry, register A contains the character to be read, and register B contains 0 for normal use or 1 for immediate action (which may cause the screen to flicker). The pattern (see re-definition of graphics characters in the Escape Sequences section of chapter 3) will be output to 12 consecutive bytes of memory. On entry, register DE must contain the address of the first byte.

CHAPTER 5

KEYBOARD HANDLING

This chapter describes keyboard handling EMT instructions.

Transfer of data from the keyboard to the CPU is controlled by the firmware, which converts keyboard strokes into 7-bit, ASCII-coded information. This has the advantage that key position changes on the keyboard layout do not cause problems when machines are updated.

In ROS, the interrupt-driven nature of keyboard operation permits the system to accept keyboard strokes and store some characters until they are required by the program.

Hardware details of the 380Z and 480Z machines are given in the respective Information File manuals, but we strongly recommend that you use EMT instructions for keyboard handling.

The first section of this chapter is a summary of the EMT instructions and version differences, and the second section gives definitions.

SUMMARY OF INSTRUCTIONS AND VERSION DIFFERENCES

Table 5.1 summarizes the keyboard handling EMT instructions.

Table 5.1 Keyboard handling EMT instructions

Mnemonic	Code (hex)	Function
KBDTL	31 (1F)	Test keyboard for depression (key stroke)
KBDW	33 (21)	Wait for character, read it, clear keyboard
KBDWF	34 (22)	As KBDW plus test for entry to Front Panel
KBDC	2 (02)	Read the keyboard, trap <CTRL/C>
KBDIN	29 (1D)	Read and clear the keyboard
KBDTC	30 (1E)	Test the keyboard, return character
KBDTF	32 (20)	Test the keyboard for entry to Front Panel

Table 5.2 gives implementation details and version differences.

Table 5.2 Implementation details and differences

Name	Code	COS 3.0	COS 3.4	COS 4.0	COS 4.2	ROS 1.0	ROS 1.1	ROS 1.2	ROS 2.2
KBDTL	31	*	●	*	*	*	*	*	*
KBDW	33	*	●	*	*	*	*	*	*
KBDWF	34	*	●	*	*	*	*	*	*
KBDC	2	*	●	*	*	*	*	*	*
KBDIN	29	*	●	*	*	+	+	+	+
KBDTC	30	*	●	*	*	*	*	*	*
KBDTF	32	*	●	*	*	x	x	x	x

● Implemented + Almost the same
* Fully compatible X Not implemented

The three instructions at the top of these tables, KBDTL, KBDW, and KBDWF, are the most useful ones. We recommend that you use these, rather than the other five, wherever possible.

KBDTL returns control to the program whether a key has been pressed or not. This is useful, since your program can find a character, if one is present, or do something else if a character is not present.

KBDW and KBDWF wait for a key to be pressed before returning to the program with the keyed-in character.

Several of the instructions involve the autopaging flag. Autopaging, and the flags involved, are discussed in chapter 2. In all versions of ROS, <CTRL/A> takes effect immediately, irrespective of any keyboard EMT instructions.

DEFINITIONS

KBDTL

Test keyboard for depression (Key stroke)

Decimal code (hex) : 31 (1F)

Implementation : All versions of COS and ROS

Registers affected : A and the flags

KBDTL reads the keyboard without clearing the character:

- If no character has been keyed in, or if <CTRL/A> has been entered, then it returns 0 in register A and sets the zero flag.
- If any character (except <CTRL/A>) has been keyed in, it returns -1 in register A and clears the zero flag.

If <CTRL/A> is entered, the autopaging flag is switched (see chapter 2).

This instruction is called using a transfer vector, except in COS 3.0 (see chapter 13 for its offset address).

KBDW

Wait for a character, read it, clear the keyboard

Decimal code (hex) : 33 (21)

Implementation : All versions of COS and ROS

Registers affected : A

KBDW waits for a character to be keyed-in. When this happens, the character is returned in register A with the flags undefined.

If <CTRL/A> is keyed in, the autopaging flag is switched (see chapter 2).

If the blink-on facility (see chapter 3) is in operation, KBDW causes the cursor to blink while it is waiting.

KBDWF

As KBDW plus test for entry to Front Panel

Decimal code (hex) : 34 (22)

Implementation : All versions of COS and ROS

Registers affected : A and the flags

KBDWF does the same as KBDW, except that it enters the Front Panel if you key in <CTRL/F>. Program context is not lost; if you leave the Front Panel with the K command, COS/ROS will re-enter the calling program.

As with KBDW, <CTRL/A> switches the autopaging

flag (see chapter 2), and the cursor will blink during a call to KBDWF if blink-on (see chapter 3) is set.
This instruction is called using a transfer vector (see chapter 13 for its offset address).

KBDC

Read the keyboard, trap <CTRL/C>

Decimal code (hex) : 2 (02)

Implementation : All versions of COS and ROS

Registers affected : A and the flags

KBDC reads the keyboard. If a character is there, it is returned in register A and the Z flag is cleared. If there is no character, 0 is returned in register A and the Z flag is set. The keyboard is cleared.

If <CTRL/C> is keyed in, it is trapped. CP/M or CP/NET is re-loaded, if present; otherwise command returns to COS/ROS.

This instruction is called using a transfer vector (see chapter 13 for its offset address).

KBDIN

Read and clear the keyboard

Decimal code (hex) : 29 (1D)

Implementation : All versions of COS and ROS

Registers affected : A and the flags

KBDIN reads the keyboard. It returns the character in register A and clears the Z flag, or, if there is no character, it returns 0 in register A and sets the Z flag. The keyboard is cleared.

In ROS, if <CTRL/A> has been keyed-in, it is not returned.

KBDTC

Test keyboard and return character

Decimal code (hex) : 30 (1E)

Implementation : All versions of COS and ROS

Registers affected : A and the flags.
KBDTC reads the keyboard without clearing it. If a character is available, it is returned in register A and the Z flag is cleared. Otherwise, 0 is returned in register A and the Z flag is set.

If <CTRL/A> is keyed in, the autopaging flag is switched (see chapter 2).

KBDTF

Test the keyboard for entry to Front Panel

Decimal code (hex) : 32 (20)

Implementation : All versions of COS

Registers affected : None

KBDTF tests for the <CTRL/F> and <CTRL/A> characters. If they have not been keyed in, KBDTF returns without clearing the keyboard, and without affecting any registers.

If <CTRL/F> is keyed in, the Front Panel is entered and the keyboard is cleared; if you leave the Front Panel with the K command, COS will re-enter the calling program.

CHAPTER 6

PRINTER AND INTERFACE HANDLING

This chapter gives details of EMT instructions that support printer and interface handling.

There is an introductory section that gives details of parallel and serial interfaces in 380Z and 480Z machines. The second section summarizes the EMT instructions and version differences. Finally, there is a definitions section.

INTRODUCTION

The 380Z and 480Z machines have hardware differences in their interfaces (see the appropriate Information File manuals), but both COS and ROS have a parallel interface, known as the user I/O port. This has input and output capability, though sufficient firmware for output only.

Table 6.1 summarizes the RS232 serial interfaces in the 380Z and 480Z.

Table 6.1 Summaries of the RS 232 serial interfaces

Interface	COS		ROS	
	Input	Output	Input	Output
SIO-1	N	Y	X	X
SIO-2	N	Y	N	Y
SIO-2B	N	Y	X	X
SIO-4	Y	Y	Y	Y
SIO-5	N	Y	X	X
SIO-6	N	Y	X	X

Y Software support N No software support
X No hardware

In the 380Z, the SIO-4, SIO-5, and SIO-6 interfaces use the I/O ports shown in Table 6.2 (see also appendix C).

Table 6.2 380Z interface I/O ports

Version		SIO-4	SIO-5	SIO-6
COS 3.0 + 3.4 (cassette)		C8	E8	48
COS 3.0 + 3.4 (disc)	F	E8	C8	48
and COS 4.0	M	C8	E8	48
COS 4.2		E4	C8	48

In COS 3.0, 3.4, and 4.0, these interfaces use an Intel 8251A chip. In COS 4.2, the SIO-5 and SIO-6 interfaces use the same type of chip, but the SIO-4 interface uses a Z80-SIO chip.

The 480Z I/O ports can be found in appendix C.

SUMMARY OF INSTRUCTION AND VERSION DIFFERENCES

Table 6.3 summarizes the printer and interface-handling EMT instructions.

Table 6.3 Printer and interface-handling EMT instructions

Mnemonic	code (hex)	Function
LPOUT	5 (05)	Output byte in reg. A to interface
OUT1	6 (06)	Spare output EMT using transfer vector
OUT2	7 (07)	As OUT1
IN1	8 (08)	Spare input EMT using transfer vector
IN2	9 (09)	As IN1
IN3	10 (0A)	As IN2
SETLST	41 (29)	Set printer from registers A and E
S4KTL	47 (2F)	Test SIO-4 interface for a character
S4KIN	48 (30)	Read SIO-4 interface into reg. A
LPSTAT	50 (32)	Check that printer is ready

Table 6.4 gives implementation details and version differences.

Table 6.4 Implementation details and differences

Name	code	COS 3.0	COS 3.4	COS 4.0	COS 4.2	ROS 1.0	ROS 1.1	ROS 1.2	ROS 2.2
LPOUT	5	*	●	*	*	*	*	*	*
OUT1	6	*	●	*	*	*	*	*	*
OUT2	7	*	●	*	*	*	*	*	*
IN1	8	*	●	*	*	*	*	*	*
IN2	9	*	●	*	*	*	*	*	*
IN3	10	*	●	*	*	*	*	*	*
SETLST	41	*	●	*	+	+	+	+	+
S4KTL	47	*	●	*	+	*	*	+	+
S4KIN	48	*	●	*	+	*	*	+	+
LPSTAT	50	X	●	*	*	*	*	*	*

● Implemented
* Fully compatible
\+ Almost the same
X Not implemented

LPOUT handles interface outputs; an 8-bit character (see the ASCII code, table 2.1 in chapter 2) is sent to an interface selected by the SETLST instruction. LPSTAT allows you to check that the selected printer, or other device, is ready.

CAUTION: If you send a control character to a printer (with the LPOUT instruction), consult your printer manual for details of control-character interpretation.

The two instructions, S4KTL and S4KIN, handle input at the SIO-4 interface. The former checks that a character is available; the latter reads and returns a 7-bit character.

DEFINITIONS

LPOUT

Output byte in register A to interface

Decimal code (hex) : 5 (05)

Implementation : All versions of COS and ROS

Registers affected : None

This instruction sends an 8-bit character (see table 2.1, chapter 2) in register A to the currently selected interface; interface selection is done with the SETLST instruction.

This instruction is called using a transfer vector (see chapter 13 for its offset address).

OUT1

Undefined output EMT linked to I/O channel

Decimal code (hex) : 6 (06)

Implementation : All versions of COS and ROS

Registers affected : Undefined

This is a spare instruction that can be defined by you.

This instruction is called using a transfer vector (see chapter 13 for its offset address).

OUT2

Undefined output EMT

Decimal code (hex) : 7 (07)

Implementaton : All versions of COS and ROS

Registers affected : Undefined

As OUT 1

IN1

Undefined input EMT linked to I/O channel

Decimal code (hex) : 8 (08)

Implementation : All versions of COS and ROS

Registers affected : Undefined

As OUT1

IN2

Undefined input EMT

Decimal code (hex) : 9 (09)

Implementation : All versions of COS and ROS

Registers affected : Undefined

As IN1

IN3

Undefined input EMT

Decimal Code (hex) : 10 (0A)

Implementation : All versions of COS and ROS

Registers affected : Undefined

As IN1

SETLST

Set printer from registers A and E

Decimal code (hex) : 41 (29)

Implementation : All versions of COS

Registers affected : None

This instruction sets up a printer or other device by connecting the LPOUT and LPSTAT instructions to the appropriate interface, and by performing any necessary initialization.

On entry, register A holds the interface code, and register E holds the baud rate code (where appropriate). Interface and baud rate codes are given in table 5.3(a) and (b), respectively.

Table 5.3(a) Interface codes

Code	Interface (in register A)
0	Monitor screen
1	SIO-1
2	SIO-2
3	parallel
4	SIO-4
5	SIO-5
6	SIO-6

Table 5.3(b) Baud rate codes

Code	Baud rate (in register E)
0	110
1	300
2	600
3	1200
4	2400
5	4800
6	9600

For example, if you want to connect a serial printer (operating at a baud rate of 4800) to the SIO-4 interface, you would put 4 in register A and 5 in register E.

Details of the baud rate limits for specific interfaces are given below:

- Interface 0 is the monitor screen which does not require a baud rate.
- The SIO-1 interface does not require a baud rate. The setting is changed in the hardware (a bank of switches on the board).
- The SIO-2 interface requires a baud rate, but it cannot operate above 2400 baud.
- The user I/O port is used for printers with a parallel interface, and a baud rate is not required.

  If an error condition exists, COS (not 4.2) will display the message:

  Attend to printer!

  Possible causes of error are:

  - Printer not switched on.
  - Printer plugged into the wrong socket.
  - Wrong type of connecting cable.
  - Printer out of paper.
  - Printer not selected by a select or online switch on the printer.

- The SIO-4, SIO-5, and SIO-6 interfaces handle baud rates of up to 9600 baud.

The default setting for SETLST is that all output (from the LPOUT instruction) goes to the monitor screen. If you press the reset button, output will be re-directed to the screen.

In COS 4.2, the SETLST instruction is called using a transfer vector (see chapter 13 for its offset address).

SETLST modification

Implementation : All versions of ROS

Registers affected : None

The interface codes, placed in register A, are restricted to 0, 2, 3, or 4.

Only ROS 1.0 displays an error message if an error condition exists with the parallel interface.

In ROS 1.2 and 2.2, the instruction is called using a transfer vector (see chapter 13 for its offset address).

S4KTL

Test SIO-4 interface for a character

Decimal code (hex) : 47 (2F)

Implementation : COS 3.0, 3.4 and 4.0; ROS 1.0 and 1.1

Registers affected : A and the flags

This instruction checks the SIO-4 serial interface, and either returns -1 (FF hex) in register A with the zero flag cleared if a character is available, or returns 0 in register A with the zero flag set. The interface must have been initialized previously, by calling the routine SETLST.

S4KTL modification

Implementation : COS 4.2, and ROS 1.2 and 2.2

The instruction is called using a transfer vector (see chapter 13 for its address)

S4KIN

Read the SIO-4 interface into register A

Decimal code (hex) : 48 (30)

Implementation : COS 3.0, 3.4 and 4.0; ROS 1.0 and 1.1

Registers affected : A and the flags

This instruction reads the SIO-4 serial interface, and returns the first available character in register A, stripping off the most significant (parity) bit.

This routine does not return to your program until a character is available; so, to prevent a possible "locking up" of the program, a call to S4KTL should be made first to ensure that a character is waiting. S4KIN assumes that the interface has been previously initialized.

S4KIN modification

Implementation : COS 4.2, and ROS 1.2 and 2.2

This instruction is now called using a transfer vector (see chapter 13 for its offset address).

LPSTAT

Check that the printer is ready

Decimal code (hex) : 50 (32)

Implementation : All versions of COS except COS 3.0. All versions of ROS.

Registers affected : A and the flags

This instruction checks that the device set up by the SETLST instruction is ready to receive a character. If it is, -1 is returned in register A; if not, 0 is returned and the zero flag is set.

The LPSTAT instruction is useful in preventing a lock up, which results when LPOUT calls a printer that is not ready.

The LPSTAT instruction is called using a transfer vector (see chapter 13 for the offset address).

CHAPTER 7

CASSETTE HANDLING

This chapter gives details of EMT instructions for cassette input and output. These instructions can be used to read and write bytes of data on cassette, and to set cassette transfer speed.

Details of data and program formats on cassette tape are provided in the 380Z and 480Z Information File manuals.

Use of a cassette recorder other than the type supplied by Research Machines can cause problems. If the recorder has phase inversion between its input and output, it will not work. Consult the Information File manuals for details.

The first section of this chapter is a summary of the EMT instructions and version differences, and the second section gives definitions.

SUMMARY OF INSTRUCTIONS AND VERSION DIFFERENCES

Table 7.1 summarizes the cassette-handling EMT instructions.

Table 7.1 Cassette-handling EMT instructions

Mnemonic	Code (hex)	Function
PUTBYT	3 (03)	Output byte in register A to tape
GETBYT	4 (04)	Read byte from tape to register A
GETSYN	17 (11)	Get synchronization character in register A from tape
SETCAS	40 (28)	Set transfer speed from register A
CASCTL	63 (3F)	Initialize the cassette system

Table 7.2 gives implementation details and version differences.

Table 7.2 Implementation details and differences

Name	Code	COS 3.0	COS 3.4/C	COS 3.4/M+F	COS 4.0	COS 4.2	ROS 1.0	ROS 1.1	ROS 1.2	ROS 2.2
PUTBYT	3	*	●	*	-	-	-	-	-	-
GETBYT	4	*	●	*	-	-	-	-	-	-
GETSYN	17	*	●	*	X	X	+	+	+	+
SETCAS	40	*	●	-	X	X	*	*	*	*
CASCTL	63	X	X	X	X	X	●	*	*	*

●	Implemented	-	Major differences
*	Fully compatible	X	Not implemented
+	Almost the same		

CASCTL was introduced with ROS to initialize and control cassette input and output operations. Consequently, COS and ROS are incompatible for such operations.

Although PUTBYT and GETBYT are present in COS 4.0 and 4.2, these two firmware versions do not support cassette operations.

Data Transfer Rate

SETCAS can be used to set the data transfer rate between fast (1200 baud) and slow (300 baud). This instruction is not fully implemented in COS 3.4/M and F. To change the transfer rate for these two versions, you must use the utility program, SCASS, which is on your CP/M operating system disc. After SCASS has been run, the transfer rate remains effective until the reset button is pressed, or until FILEX or SCASS is run again.

Note that the FILEX program contains its own slow handler for the cassette system, so SCASS is not necessary in this case.

DEFINITIONS

PUTBYT

Output the byte in register A to tape

Decimal code (hex) : 3(03)

Implementation : COS 3.0 and 3.4

Registers affected : None

PUTBYT outputs the character in the A register to tape. The character is output immediately with no imposed file structure; this means that it will be difficult to find a given character when reading from tape.

This instruction is called using a transfer vector, (see chapter 13 for its offset address).

PUTBYT modifications

Implementation : COS 4.0 and COS 4.2

Because these versions do not support cassette operations, the transfer vector is initialized to break to the Front Panel.

Implementation : All versions of ROS

CASCTL must be called before any call to PUTBYT.

GETBYT

Read byte from tape into register A

Decimal code (hex) : 4 (04)

Implementation : COS 3.0 and 3.4

Registers affected : A and the flags

GETBYT returns the next character read from the tape into register A. The instruction returns information when a character has been read from the tape, or when a control character is entered at the keyboard.

A character from tape is returned in register A, and the carry flag is cleared. For a keyboard-entered character, the carry flag is set and A is undefined.

This instruction is called using a transfer vector (see chapter 13 for its offset address).

GETBYT modifications

Implementation : COS 4.0 and 4.2

Because these versions do not support cassette operation, the transfer vector is initialized to break to the Front Panel.

Implementation : All versions of ROS

CASCTL must be called before any call to GETBYT.

GETSYN

Get synchronization character from tape

Decimal code (hex) : 17(11)

Implementation : COS 3.0 and 3.4. All versions of ROS

Registers affected : The flags

GETSYN searches for a synchronization character which marks the beginning of a block of data on tape. Register A must contain the synchronization character before entry to the instruction.

First, a search is made for an inter-record gap (240 cycles of 2.4 kHz tone, or 100 msec of no recorded information). On finding a gap, the next character is compared with the synchronization character. The instruction returns to the calling program if a match occurs. Otherwise the search for an inter-record gap continues.

This procedure accurately defines the beginning of a block of information and the tape can be started and read from rest, with no generation of erroneous characters.

For BASIC, BASLOAD, BASDUMP, TXED, and FILEX, the cassette file system format uses 16 hex as the synchronization character; COS and ROS use 4D hex.

While GETSYN is in operation, the keyboard is constantly monitored. You can abort in COS 3.0 and 3.4 by keying a control character, and in ROS 1.2 and 2.2 by entering <CTRL/Z>. You cannot abort in ROS 1.0 and 1.1. If you abort, GETSYN returns to the calling program with the carry flag set. Otherwise, the carry flag is reset when GETSYN exits.

This instruction is called using a transfer vector (see chapter 13 for its offset address).

SETCAS

Set data transfer speed from register A

Decimal code (hex) : 40 (28)

Implementation : COS 3.0 and 3.4. All versions of ROS

Registers affected : A

SETCAS is used to set the data transfer speed between fast (1200 baud) and slow (300 baud). The two least significant bits in register A control the speed of input and output. Bit 0 (least significant) is set for fast input and reset for slow input. Bit 1 (next least significant) is set for fast output, and reset for slow output.

On entry, bits 0 and 1 of register A hold the desired speed settings. The old settings are returned in A. In COS 3.4/M or F, the value 3 is always returned because both input and output are fast.

CASCTL

Initialize the cassette system

Decimal code (hex) : 63 (3F)

Implementation : All versions of ROS

Registers affected : None

The cassette system is initialized by CASCTL. It is not necessary to call CASCTL before every invocation of GETBYT or PUTBYT; but you must call CASCTL before you start.

When you have finished cassette operations, call CASCTL with register A containing 0 (failure to do so will disable any currently-selected hardware and software printer option, and machine operation will be slowed down).

If you have a dual cassette controller, the initialization by CASCTL is:

- Before using GETBYT, register A must contain 1. This switches on the appropriate cassette controller.

- Before using PUTBYT, register A must contain 2. Again, the appropriate motor is switched on.

You can abort a tape read or write operation by entering <CTRL/SHIFT/8> or <CTRL/SHIFT/9>. This activates an automatic call to CASCTL to stop the cassette recorder.

CHAPTER 8

DISC HANDLING

This chapter describes disc-handling EMT instructions. These instructions can be used to read and write sectors of data on a disc. However, you must never use these EMT instructions under CP/M.

There are several forms of disc system; these are summarized in Table 8.1.

Table 8.1 Summary of disc systems

Firmware Version	Disc Systems
COS 3.0/M COS 3.0/F COS 3.4/M COS 3.4/F COS 4.0/M COS 4.0/F (380Z)	5.25-inch diameter (Mini) or 8-inch diameter (Full) discs. Single (data) density. Floppy disc controller (FDC) board.
COS 4.2 (380Z-D)	5.25-inch diameter or 8-inch diameter discs. Double (data) density. Quad density(5.25-inch only). Intelligent disc controller (IDC) board.
ROS 1.2 ROS 2.2 (480Z)	5.25-inch diameter discs. Double and quad (data) density. Intelligent disc controller (IDC) board.

The sign-on message for COS 3.0, 3.4 and 4.0 displays either the suffix, M, denoting a mini system (5.25-inch diameter discs), or F, denoting a full (8-inch diameter discs) system. In COS 4.2, there are no suffixes for disc size.

The 480Z uses only 5.25-inch diameter discs.

Other differences between the systems are outlined in the introduction section; this is followed by a section that summarizes the FDC board EMT instructions and the IDC board EMT instructions. Finally there is a definitions section.

INTRODUCTION

Systems using an FDC board are considered first, then systems using an IDC board.

FDC Board Systems

These systems (see table 8.1) use single (data) density discs. Details of tracks, sectors and data space for each disc size are given in table 8.2.

Table 8.2 Single-density disc details

	Disc Diameter	
	5.25-inch	8-inch
Tracks per side	40 (numbered 0 to 39)	77 (numbered 0 to 76)
Sectors per track	16 (numbered 1 to 16)	26 (numbered 1 to 26)
Bytes per sector	128	128
Data space per side	80 Kbytes	250 Kbytes

The FDC board EMT instructions read or write one sector at a time on single-density discs.

IDC Board Systems

These systems (see table 8.1) use double (data) density discs. Table 8.3 gives details of tracks, sectors, and data space for each double-density disc size.

Table 8.3 Double and quad-density disc details

	Disc Diameter	
	5.25 inch	8 inch
Tracks per side	Double: 40 (numbered 0 to 39) Quad: 80 (numbered 0 to 79)	77 (numbered 0 to 76)
Sectors per track	9 (numbered 1 to 9)	26 (numbered 1 to 26)
Bytes per sector	512	256
Data space per side	180 Kbytes (double) 360 Kbytes (quad)	500 Kbytes

There are 11 IDC EMT instructions that allow physical and logical read and write operations on single or double-density discs. To understand logical operations, physical and logical sectors must be defined:

- A physical sector : A portion of track on the disc surface that is accessed during a read/write operation (the sectors in tables 8.2 and 8.3)
- A logical sector: A block of 512 bytes of data is read/written regardless of disc size or density (that is, regardless of physical sector size); each block is sub-divided into four 128-byte units in a buffer on the IDC board, and each of these data units is known as a logical sector. See figure 8.1 below.

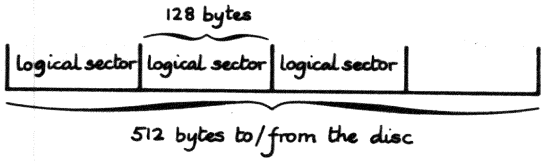


Figure 8.1 Logical sectors

Consider a logical-read operation on an 8-inch diameter, double-density disc. The physical sector is 256 bytes (from table 8.3), but the logical operation reads 512 bytes (2 physical sectors). These 512 bytes are split into 4 logical sectors. Consequently, for this type of disc, there are two logical sectors per physical sector.

The connection between the two types of sector on any type of disc can be worked out in this way. Table 8.4 summarizes the number of physical and logical sectors on each type of disc.

Table 8.4 Physical and logical sector numbers

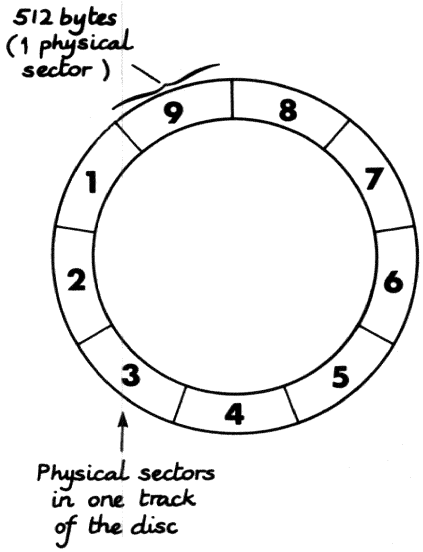
	Number of Sectors			
	5.25-inch diameter discs		8-inch diameter discs	
	Single-density	Double or quad density	Single-density	Double-density
Physical Sector	16	9	26	26
Logical Sector	16	36	26	52

Appendix B gives further details of logical sectors and disc formats.

Logical Operations

To show the relationship between physical and logical sectors, we will take the case of a logical-read operation on a 5.25-inch diameter, double-density disc, see Figure 8.2.

If you are reading the file sequentially, the first read-operation takes 512 bytes, or 1 physical sector. The 512 bytes are split into 4 logical sectors.



The 512 bytes of the first physical sector are put in a buffer of the IDC board:

1
2
3
4

These are now logical sectors 1 to 4.

1
2
3
4

Your program asks for logical sector 1. This sector is copied from the buffer.

1
2
3
4

Your program asks for logical sector 2. As this is in the buffer, there is no need to read data from the disc. Logical sector 2 is copied from the buffer.

Figure 8.2 Logical-read operation

If, however, your program now asks for logical sector 5, the IDC notes that it is not in the buffer. Consequently, another buffer is needed to read in the next 512 bytes (physical sector 6 in figure 8.2).

The buffers are maintained by a list of pointers in an order roughly corresponding to "least recently used". When a buffer is accessed it is marked "most recently used".

When all sectors from a buffer have been accessed (either read or write), the buffer is written to disc (if a write operation), then made into the "least recently used". This buffer would be the one most likely to be used next time.

If no "clean" buffers are available when needed, the "least recently used" one will be used, and (if necessary) its contents will be flushed to disc.

Flushing is the act of clearing a buffer that has write-data in it. This 512-byte block of data in the buffer is written to the disc (in its original data density) at the specified logical sector number positions.

Precautions To Be Taken When Using Logical Operations

It is not difficult to see that, if data is left in buffers, there is a danger of flushing on to the wrong disc. If the disc is changed, and then buffers are flushed, two problems will occur:

- The wrong data will be written on to the new disc, overwriting data on that disc.

- The right data will not be written on the old disc.

To avoid this kind of event, always use the EMT instruction FLUSH before you change discs. But if you have changed discs, never then use EMT instruction FLUSH; this would be catastrophic.

It is also good practice to use the EMT INISYS before any logical operations on a newly inserted disc, because this EMT will initialize the buffering system on the IDC board so that there is no valid data remaining in the buffers.

NOTE: With logical operations, sequential data access is very fast; if you use random access, data access is not so fast.

Sector Parameter and Information Addressing

Both the FDC and the IDC EMT instructions require that you place information (sector parameters, read/write information) in RAM (see the definitions section). You should make sure that you have allotted sufficient memory space for the data to be handled.

Information is handled in sector units, but, as you will have seen from the previous sections, these units are not standard. Logical operations require least memory for information (128 bytes); physical operations require 128 bytes for single-density discs, but 256 or 512 bytes for double-density discs.

SUMMARY OF INSTRUCTIONS AND VERSION DIFFERENCES

Table 8.5 summarizes the disc-handling EMT instructions, and table 8.6 gives implementation details plus version differences.

Table 8.5 Disc EMT routines

	Mnemonic	Code (hex)	Function
FDC INSTRUCTIONS	INIT	25 (19)	Initialize disc drive
	RDSEC	26 (1A)	Read single-density, physical sector
	WRSEC	27 (1B)	Write single-density, physical sector
	WRCHK	28 (1C)	Write & check single-density, physical sector
	BOOT	49 (31)	Perform a cold bootstrap
IDC INSTRUCTIONS	INISYS	64 (40)	Initialize disc system
	RDSECP	65 (41)	Read physical disc sector
	WRSECP	66 (42)	Write physical disc sector
	WRCHKP	67 (43)	Write & check physical sector
	RDSECL	68 (44)	Read logical sector
	WRSECL	69 (45)	Write logical sector
	WRCHKL	70 (46)	Write & check logical sector
	FLUSH	71 (47)	Flush buffers
	RDINFO	72 (48)	Read disc drive information
	FORMAT	73 (49)	Format disc track
	VERTRK	74 (4A)	Verify format of disc track

Table 8.6 Implementation details and version differences

Name	Code	COS 3.0	COS 3.4	COS 4.0	COS 4.2	ROS 1.0	ROS 1.1	ROS 1.2	ROS 2.2
INIT	25	*	●	*	+	-	-	*	*
RDSEC	25	*	●	*	+	-	-	+	+
WRSEC	27	*	●	*	+	-	-	+	+
WRCHK	28	*	●	*	+	-	-	+	+
BOOT	49	*	●	*	+	-	-	*	*
INISYS	64	X	X	X	●	X	X	*	*
RDSECP	65	X	X	X	●	X	X	*	*
WRSECP	66	X	X	X	●	X	X	*	*
WRCHKP	67	X	X	X	●	X	X	*	*
RDSECL	68	X	X	X	●	X	X	*	*
WRSECL	69	X	X	X	●	X	X	*	*
WRCHKL	70	X	X	X	●	X	X	*	*
FLUSH	71	X	X	X	●	X	X	*	*
RDINFO	72	X	X	X	●	X	X	*	*
FORMAT	73	X	X	X	●	X	X	*	*
VERTRK	74	X	X	X	●	X	X	*	*

●	Implemented	+	Almost the same
*	Fully compatible	X	Not implemented
-	Major differences		

With COS 3.0, 3.4 and 4.0, the FDC instructions will access an FDC board to read and write single-density data. With COS 4.2, FDC instructions will access the IDC board and use single-density data transfers.

IDC instructions (which do not exist in FDC-based systems) will read and write discs in single or double data density. IDC instructions are faster than FDC instructions.

All instructions, except BOOT in COS versions, are called using transfer vectors (see chapter 13 for the offset addresses).

DEFINITIONS

The first two sections provide definitions of the FDC EMT instructions and the IDC EMT instructions, respectively. Error and exceptional condition codes, common to all IDC instructions, are defined in the third section.

FDC EMT Definitions

INIT

Initialize disc unit

Decimal Code (hex) : 25 (19)

Implementation : All versions of COS;

Registers affected : A and the flags

INIT initializes a specific disc drive, the unit number of which is held in the memory location addressed by register IX.

On return, if the call is successful, register A will contain 0 and the zero flag will be set.

If unsuccessful, a bit is set in register A, signalling an error:

- bit 7 = Not ready
- bit 4 = No track 00 signal

Versions earlier than COS 4.2 fold the drive number into the range 0 to 3.

INIT modification

Implementation : ROS 1.2 and 2.2

An additional error can occur. If the parameter is out of range, the zero flag is clear, the carry flag is set, and bit 5 is set.

RDSEC

Read sector

Decimal code (hex) : 26 (1A)

Implementation : All versions of COS; ROS 1.2 and 2.2

Registers affected : A and the flags

RDSEC reads a sector from a 48 track per inch (tpi) disc off either a 48 tpi or a 96 tpi drive.

The parameters of the sector to be read are contained in a 5-byte block of memory, as shown below. The address in register IX points to the first of these bytes:

```
IX =>  DEFB UNIT       (1 byte)
       DEFB TRACK      (1 byte)
       DEFB SECTOR     (1 byte)
       DEFW ADDRESS    (2 bytes)
```

The first byte, labelled UNIT, specifies the disc drive (A, B, C, or D) to be accessed:

00 hex = A

01 hex = B

02 hex = C

03 hex = D

The second byte, labelled TRACK, specifies the track number. With an FDS system, the range is 0 to 76 (0 to 4C hex); with an MDS system the range is 0 to 39 (0 to 27 hex).

The third byte, labelled SECTOR, specifies the sector number. It ranges from 1 to 26 (1 to 1A hex) with an FDS system, and 1 to 16 (1 to 10 hex) with an MDS system.

The 2 bytes labelled ADDRESS specify the memory address at which the data read from the sector will be stored.

On exit, if no error has occurred, register A contains 0 and the zero flag will be set. If an error has occurred, a bit is set in register A:

- Bit 7 = Drive not ready
- Bit 4 = Sector not found/seek error
- Bit 3 = CRC error
- Bit 2 = Data lost

RDSEC modification

Implementation : ROS 1.0 and 1.1

The transfer vector used for calling this instruction is initialized to break to Front Panel.

WRSEC

Write sector, no check

Decimal code (hex) : 27 (1B)

Implementation : All versions of COS; ROS 1.2 and 2.2

Registers affected : A and the flags

WRSEC writes to a specified sector on a 48 tpi disc (with 48 or 96 tpi drives).

The parameters are defined in the same way as described in RDSEC definition, except that the two ADDRESS bytes contain the memory address at which the data to be written is stored.

On exit, if no error has occurred, register A contains 0 and the zero flag is set. If an error has occurred, a bit is set in register A:

- Bit 7 = Drive not ready
- Bit 6 = Disc is write protected
- Bit 5 = Write fault
- Bit 4 = Sector not found/seek error
- Bit 3 = CRC error
- Bit 2 = Data lost

WRSEC modification

Implementation : ROS 1.0 and 1.1

The transfer vector used for calling this instruction is initialized to break to Front Panel.

WRCHK

Write and check sector

Decimal code (hex) : 28 (1C)

Implementation : All versions of COS; ROS 1.2 and 2.2

Registers affected : A and the flags

WRCHK writes out the specified sector, exactly as for WRSEC, then reads it back for verification. On exit, if no error has occurred, register A holds 0 and the zero flag is set. Error codes, returned as usual in A, are:

- Bit 7 = Drive not ready
- Bit 6 = Disc is write protected
- Bit 5 = Write fault
- Bit 4 = Sector not found/seek error
- Bit 3 = CRC error
- Bit 2 = Data lost
- Bit 1 = Data mismatch

WRCHK Modification

Implementation : ROS 1.0 and 1.1

The transfer vector used for calling this instruction is initialized to break to Front Panel.

BOOT

Initiate a cold bootstrap

Decimal code (hex) : 49 (31)

Implementation : All versions of COS; ROS 1.2 and 2.2

BOOT puts the system into a defined state, resets the stack pointer, and tries to load

track 0, sector 1 into memory at 80 hex. In IDC systems, EMT RDSECL is used to read the sector, and although only the first sector is loaded, sectors 1,2, 3, and 4 must exist, otherwise a seek error will occur.

When the firmware commands, B or X, are entered, BOOT searches on drive A or B, respectively.

In ROS versions, BOOT is called using a transfer vector (see chapter 13 for its offset address).

BOOT modification

Implementation : ROS 1.0 and 1.1

The transfer vector used for calling this instruction is initialized to break to Front Panel.

IDC EMT Definitions

Errors returned from IDC EMT instructions are described on page 8.19.

INISYS

Initialize disc system

Decimal code (hex) : 64 (40)

Implementation : COS 4.2, ROS 1.2 and 2.2

Registers affected : A and the flags

INISYS is used to select a disc drive for booting the operating system. INISYS also sets the maximum number of retries (of reading or writing operations) to be attempted if errors occur.

On entry to the instruction, register IX must point to a byte in memory containing:

- Bits 4 to 7 specify the number of retries to be attempted (0 to 14). If this number is 15, there is no change to the currently specified number of retries.

- Bit 3 = Drive change

  0 signifies change the drive mapping, clear all buffers.
  1 signifies no change to the drive mapping.

- Bits 2 to 0 = Drive number (0 to 7) of logical drive A for booting

This instruction destroys the contents of all buffers in the IDC, then initializes them.

Note that the error-retry mechanism only operates with logical operations and with INIT.

RDSECP

Read physical sector

Decimal code (hex) : 65 (41)

Implementation : COS 4.2, ROS 1.2 and 2.2

Registers affected : A and the flags

RDSECP reads a physical sector (without buffering).

The parameters of the sector to be read are contained in a 5-byte block of memory, as shown below. The address in register IX points to the first of these bytes:

```
IX =>     DEFB  UNIT      (1 byte)
          DEFB  TRACK     (1 byte)
          DEFB  SECTOR    (1 byte)
          DEFW  ADDRESS   (2 bytes)
```

The first byte, labelled UNIT, specifies the following:

- Bit 7 = Reserved. Should be 0.
- Bit 6 = Track density indicator.
  0 is 48 tpi.
  1 is 96 tpi.

A 96 tpi request will be forced to 48 tpi on a 48 tpi drive unit.

- Bit 5 = Data density. If 1, double density. If 0, single density.
- Bit 4 = Reserved. It should be 0.
- Bit 3 = If 1, no disc access is made and the data held on the IDC is used. If 0, disc access is made.
- Bits 2 to 0 = Drive number (0 to 7) Data access.

The second byte, labelled TRACK signifies the number of the track that holds the sector to be read. This number ranges from 0 to 76 (0 to 4C hex) on FDS systems and 0 to 39 (0 to 27 hex) on MDS systems.

The third byte, labelled SECTOR, signifies the sector number; it is in one of the following ranges:

- 5.25-inch single-density = 1 to 16 (1 to 10 hex)
- 5.25-inch double or quad density = 1 to 9 (1 to 9 hex)
- 8-inch single & double-density = 1 to 26 (1 to 1A hex)

The two bytes labelled ADDRESS specify the memory address at which the data read from the sector will be stored.

WRSECP

Write physical sector

Decimal code (hex) : 66 (42)

Implementation : COS 4.2, ROS 1.2 and 2.2

Registers affected : A and the flags

WRSECP writes a physical sector (without buffering).
The memory block parameters are the same as described in the RDSECP definition except for bit 3 in the UNIT parameter.

WRCHKP

Write and check physical sector

Decimal code (hex) : 67 (43)

Implementation : COS 4.2, ROS 1.2 and 2.2

Registers affected : A and the flags

WRCHKP writes a physical sector (without buffering). The disc controller reads it back and checks it.

The memory block parameters are the same as described in the RDSECP definition.

RDSECL

Read logical sector

Decimal code (hex) : 68 (44)

Implementation : COS 4.2, ROS 1.2 and 2.2

Registers affected : A and the flags

RDSECL reads a logical sector (using buffering).

The memory block parameters are the same as described in the RDSECP definition except for the range of the SECTOR byte.

The SECTOR byte is in one of the following ranges:

- 5.25-inch single-density = 1 to 16 (1 to 10 hex)
- 5.25-inch double or quad density = 1 to 36 (1 to 24 hex)
- 8-inch single-density = 1 to 26 (1 to 1A hex)
- 8-inch double-density = 1 to 52 (1 to 34 hex)

WRSECL

Write logical sector

Decimal code (hex) : 69 (45)

Implementation : COS 4.2, ROS 1.2 & 2.2

Registers affected : A and the flags

WRSECL writes a logical sector (using buffering).

The memory block parameters are the same as described in the RDSECL definition.

WRCHKL

Write and check logical sector

Decimal code (hex) : 70 (46)

Implementation : COS 4.2, ROS 1.2 & 2.2

Registers affected : A and the flags

WRCHKL writes a logical sector to the disc (using buffering). The disc controller reads it back and checks it.

The memory block parameters are the same as described in the RDSECL definition.

FLUSH

Flush Buffer

Decimal code (hex) : 71 (47)

Implementation : COS 4.2, ROS 1.2 & 2.2

Registers affected : A and the flags

FLUSH flushes all buffers that are associated with the specified disc drive, and that contain valid data not yet written to the disc. This is done in the same density as that in which of the data was read. The IX register points to a UNIT byte with the following format:

- Bits 7 to 4 = Ignored
- Bit 3 = Specify either all buffers (1), or only the buffers on the specified drive (0).
- Bits 2 to 0 = Specify the disc drive

RDINFO

Read disc drive information

Decimal code (hex) : 72 (48)

Implementation : COS 4.2, ROS 1.2 & 2.2

Registers affected : A and the flags

RDINFO obtains information about any connected disc drive.

On entry, the X register must point to a valid UNIT byte (see the FLUSH description for its format).

On successful exit, the Z flag is set and the A register contains:

- Bits 7 to 4 = Reserved for future use. Currently 0.
- Bit 3 = 0 if 48 tpi drive or 1 if 96 tpi drive.
- Bit 2 = 0 if 48 tpi disc or 1 if 48 tpi disc.
- Bit 1 = 0 if single-density or 1 if double-density.
- Bit 0 = 0 if 5.25-inch or 1 if 8-inch.

FORMAT

Format disc track

Decimal code (hex) : 73 (49)

Implmentation : COS 4.2, ROS 1.2 & 2.2

Registers affected : A and the flags

FORMAT can be used by utilities that format floppy discs. A safety feature that prevents disc damage has been incorporated. One of the formatting parameters must point to a valid check byte for formatting to take place.

The parameters of the track to be formatted are contained in a 5-byte block of memory, as shown below. The address in register IX points to the first of these bytes:

```
IX =>   DEFB   UNIT          (1 byte)
        DEFB   TRACK         (1 byte)
        DEFB   0             (1 byte)
        DEFW   ADDRESS       (2 bytes)
```

The first byte, labelled UNIT, specifies the required disc density:

- Bit 7 = Reserved. Should be 0.
- Bit 6 = Track density indicator.

  0 = 48 tpi.
  1 = 96 tpi.
- Bit 5 = Data density:
  0 is single-density
  1 is double-density
- Bit 3 = Set to 0
- Bits 2 to 0 = Disc drive number (0-7)

The second byte, labelled TRACK, signifies the track number to be formatted. This ranges from 0 to 76 (0 to 4C hex) on FDS systems and 0 to 39 (0 to 27 hex) on MDS systems.

The two bytes labelled ADDRESS must point to a valid check byte, whose contents depend on the size of drive and the data density:

- 5.25-inch, single-density = 16 (10 hex)
- 8-inch, single-density = 26 (1A hex)
- 5.25-inch, double or quad density = 9 (9 hex)
- 8-inch, double-density = 26 (1A hex)

These controls are, in fact, the number of physical sectors per track.

On successful exit the Z flag is set, and the A register the values shown in the RDINFO definition.

VERTRK Verify format of a disc track

Decimal code (hex) : 74 (4A)

Implementation : COS 4.2, ROS 1.2 & 2.2

Registers affected : A and the flags

VERTRK is used in conjunction with the FORMAT EMT to check that a valid recording of the format of a disc track has been made. The memory block parameters and retrieval information are the same as described in the FORMAT definition.

IDC EMT Error Codes and Exceptional Conditions

The IDC EMT instructions return error codes and other information in register A. The states of the zero and carry flags indicate whether an error has occurred or whether an instruction has been successfully completed.

If the zero flag is not set, an error has occurred and the state of carry flag determines what type of error has occurred. If the zero flag is set, the instruction has been successfully completed.

The register A codes are summarized below:

- zero flag clear, carry flag set indicates that an error has occurred while the firmware was sending the instruction or parameters to the IDC.

  The error bits (set is significant) in register A are:

bit 7	=	drive not ready
bit 6	=	command unknown to the IDC
bit 5	=	one or more parameters out of range
bits 4 to 0	=	not used, defined as 0

- zero flag clear, carry flag clear indicates that a disc error has occurred.

  The error bits (set is significant) in register A are:

bit 7	=	Drive not ready
bit 6	=	Disc is write-protected
bit 5	=	Unknown disc error
bit 4	=	Sector not found/seek error
bit 3	=	Cyclic redundancy check (CRC) error
bit 2	=	Not used, defined as 0
bit 1	=	Verification error during a write/check EMT
bit 0	=	The error refers to a previous sector when flushing a buffer

- zero flag set, carry flag set or clear indicates a successful completion of the instruction.

  The value returned in register A depends upon the operation:

  a) disc access: 0 indicates success with no retries; 1 to 14 indicates sucess after 1 to 14 retries, and 15 indicates success after 15 or more retries (only if several sectors are accessed).

  b) during reading of a physical sector: 0 indicates a 128-byte sector, 1 indicates a 256-byte sector, and 3 indicates a 512-byte sector.

  c) during an operation which does not involve access to a disc: register A will contain information about the disc drive as described in the RDINFO definition.

CHAPTER 9

MISCELLANEOUS EMT INSTRUCTIONS

This chapter gives details of several useful EMT instructions that are not covered in the previous chapters.

The first section is a summary of the instructions and the version differences; the second section gives definitions.

SUMMARY OF INSTRUCTIONS AND VERSION DIFFERENCES

Table 9.1 summarizes the miscellaneous EMT instructions.

Table 9.1 The miscellaneous EMT instructions

Mnemonic	Code (Hex)	Function
ERROR	-1 (FF)	Display error message, then return to firmware command level
CONTC	0 (00)	Return to command level
WAIT	16 (10)	Wait B * 833 microseconds
UPDATE	18 (12)	Copy MASK to PORT0
GETHEX	19 (13)	Place a hex number from keyboard in reg. HL
DEOUT	20 (14)	Put reg. DE to (HL) as hex
BYTEO	21 (15)	Put reg. A to (HL) as hex
CHAN	24 (18)	Execute EMT given in reg. C
SAVE	35 (23)	Save regs. HL, DE and BC on stack
SAVEA	36 (24)	Save regs. Hl, DE, BC and AF on stack
FSTLST	37 (25)	Input 2 hex numbers prompted by "First" and "Last"
GETJP	38 (26)	Return Ath entry from table at (HL)
BASE	39 (27)	Return workspace address in reg. DE
VERSN	51 (33)	Return version number of firmware
MOVBLK	61 (3D)	Copy a block of memory to another area
RAMMAP	62 (3E)	Find amount of available memory

Table 9.2 gives implementation details and version differences.

Table 9.2 Implementation details

Name	Code	COS 3.0	COS 3.4	COS 4.0	COS 4.2	ROS 1.0	ROS 1.1	ROS 1.2	ROS 2.2
ERROR	-1	*	●	*	*	*	*	*	*
CONTC	0	*	●	*	*	*	*	*	*
WAIT	16	*	●	+	+	+	+	+	+
UPDATE	18	*	●	*	*	X	X	X	X
GETHEX	19	*	●	+	+	+	+	+	+
DEOUT	20	*	●	*	*	*	*	*	*
BYTEO	21	*	●	*	*	*	*	*	*
CHAN	24	*	●	*	*	*	*	*	*
SAVE	35	*	●	*	*	*	*	*	*
SAVEA	36	*	●	*	*	*	*	*	*
FSTLST	37	*	●	-	-	-	-	-	-
GETJP	38	*	●	*	*	*	*	*	*
BASE	39	*	●	*	*	*	*	*	*
VERSN	51	X	●	*	*	*	*	*	*
MOVBLK	61	X	X	X	X	●	X	X	X
RAMMAP	62	X	X	X	X	●	+	+	+

● Implemented
* Fully compatible
\+ Almost the same
\- Major differences
X Not implemented

DEFINITIONS

ERROR

Print error message, return to firmware command level

Decimal code (hex) : -1 (0FF)

Implementation : All versions of COS and ROS

Registers affected : Irrelevant

ERROR displays an error message on the screen and returns control to the firmware command level.

CONTC

Return to firmware command level

Decimal code (hex) : 0 (00)

Implementation : All versions of COS and ROS

Registers affected : Irrelevant

CONTC transfers control to the firmware command level.

WAIT

Wait B multiplied by 833 microseconds

Decimal code (hex) : 16 (10)

Implementation : All versions of COS and ROS

Registers affected : None

WAIT waits an interval before returning to the calling program. The interval time is specified by the contents of register B multiplied by 1/1200 seconds. If B contains 0, there is an interval of 256/1200 seconds in COS 3.0 and 3.4, and ROS 1.2, and 2.2; in all other versions, there is no interval.

In COS 3.0 and 3.4, this instruction uses the hardware "1200 Hz" clock. All later COS and ROS versions use a software simulation.

UPDATE

Copy MASK to PORT0

Decimal code (hex) : 18 (12)

Implementation : All versions of COS

Registers affected : None

A number of functions on the 380Z are controlled by the bit pattern written into RAM location PORT0 (see chapter 12). This is a write-only location, and any attempt to read it will return the currently available character from the keyboard.

A record of the current state of the bits in PORT0 is kept at location MASK in the workspace area of memory (0FF03H). UPDATE reads the mask word at this location and the contents are

written to PORT0.

GETHEX

Get a hex. number from the keyboard into HL

Decimal code (hex) : 19 (13)

Implementation : COS 3.0 and 3.4

Registers affected : A, B, C, HL, and the flags

GETHEX is used to obtain a hexadecimal value from the keyboard. Up to four hexadecimal digits can be input, and the entered value must be terminated by any non-hexadecimal character. A leading 0 is automatically inserted; if no hexadecimal digits are entered, 0 is returned.

Register HL contains the entered hexadecimal value converted to binary, with the most significant byte in H and the least significant byte in L. Register C contains the number of characters entered (0 to 4). Register B contains the terminating character. Register A is undefined.

If you enter <CTRL/B> during execution of this instruction, input is stopped and you return to the firmware command level. Entering <CTRL/C> reloads CP/M.

Front Panel commands requiring hexadecimal input use GETHEX; the screen format displayed when you enter, for example, <J> in Front Panel mode is the prompt that occurs with this instruction.

GETHEX modification

Implementation : COS 4.0 and 4.2
All versions of ROS

If you enter <CTRL/B> or <CTRL/C>, they act like any other terminating character, with the corresponding ASCII value left in register B.

DEOUT

Put register DE to (HL) as hex

Decimal code (hex) : 20 (14)

Implementation : All versions of COS and ROS

Registers affected : Registers A and F are destroyed. Register

HL is incremented as described below.

DEOUT is used to convert a 16-bit binary value, in register pair DE, to a hexadecimal string at the address contained in register pair HL.

On entry, HL can point to any valid memory address. Four hexadecimal characters are output to this address and the three following locations.

On return, HL points to the location following that of the last character output. Leading zeros are inserted if appropriate. The contents of DE are unchanged.

DEOUT can be used whenever binary to hexadecimal conversion is required.

BYTEO

Put register A to (HL) as hex digits

Decimal code (hex) : 21 (15)

Implementation : All version of COS and ROS

Registers affected : Register A and the flag register are destroyed. Register HL is incremented as described below

Like DEOUT, BYTEO converts a binary value to a hexadecimal string but this time converting the 8-bit binary value in register A. Two hexadecimal characters are stored at the address given in register HL and at the location following that address; on return, HL points to the location following that of the last character stored. A leading zero is inserted, if appropriate.

CHAN

Execute the EMT given in register C

Decimal code (hex) : 24 (18)

Implementation : All versions of COS and ROS

Registers affected : Depends on the EMT executed

Using this instruction, the EMT called can be

made dependent upon a previous event in the program. This is done by passing the hexadecimal code of the required EMT to CHAN in register C. Other registers should be set according to the chosen EMT.

SAVE

Save registers HL, DE, and BC

Decimal code (hex) : 35 (23)
Implementation : All versions of COS and ROS

Registers affected : HL

SAVE registers HL, DE, and BC onto the stack together with the address of a routine to pull them off again.

The following example program causes the values of these three registers to be preserved across the call to the routine labelled SUBR:

```
Subr:   push    hl
        push    de
        push    bc
        .
        .
        .
        pop     bc
        pop     de
        pop     hl
        ret
```

The program below does exactly the same:

```
Subr:   emt     save
        .                   ;HL not available
        .                   ;at this point.
        .
        ret
```

A drawback of the SAVE instruction is that, although the value contained in HL is preserved across the called routine, the old value is not preserved across SAVE; this means that the old value in HL is not available to the called routine.

SAVEA

Save registers AF, HL, DE, and BC

Decimal code (hex) : 36 (24)

Implementation : All versions of COS and ROS

Registers affected : HL

SAVEA is similar to SAVE, except that register A and the flags are also pushed on to the stack. Again, the value in HL is not available to the called routine, but is preserved across it.

FSTLST

Input first and last 16-bit numbers

Decimal code (hex) : 37 (25)

Implementation : COS 3.0 and 3.4

Registers affected : HL, DE, BC, A, and the flags

FSTLST obtains two 16-bit hexadecimal numbers from the keyboard. FSTLST prompts:

First>

and calls GETHEX to obtain the first (lesser) number. A number must now be entered and terminated by pressing any key. FSTLST then prompts:

Last>

and obtains a second (greater) hexadecimal number, again using GETHEX. On exit, the first number is held in register pair DE, and the last is in BC. Register pair HL contains the difference between these two quantities (the second minus the first). Register A and the flags are destroyed.

If the first number is greater than the second, the firmware displays the message:

?Err?

or a similar message, and control returns to the firmware command level.

FSTLST modification

Implementation : COS 4.0, 4.2 and all versions of ROS

FSTLST behaves in the same way as above except that only <RETURN>, <space>, or <CTRL/F> will terminate input.
If <RETURN> or <space> are used for termination, the instruction operates as above but with the CY flag cleared.

If <CTRL/F> is used, the sequence is aborted, the contents of the registers are unpredictable, and CY is set to indicate the situation.

Other non-hexadecimal characters are ignored, except for <DELT> which deletes the most recently entered character.

If the first number is greater than the last number, an error message is displayed, and the sequence is restarted.

GETJP

Return the Ath entry from a table at (HL)

Decimal code (hex) : 38 (26)

Implementation : All versions of COS and ROS

Registers affected : HL

GETJP is used to return an entry from a table of two-byte quantities, often addresses. The value in register A controls which entry will be returned.

For example, the following program fragment performs a "computed GOTO":

```
          .
          .
          ld      a,index     ;Select entry.
          ld      hl,table    ;Table of pointers.
          emt     getjp       ;Get entry.
          jp      (hl)        ;Go to routine.
          .
          .

table:    defw    sub1        ;Continuation of
          defw    sub2        ;main program.
          defw    sub3
          .
          .
```

On entry to GETJP register pair HL holds the address of the start of the table (zeroth entry), and register A holds the entry number. On exit, HL holds the Ath entry in the table.

BASE

Return workspace address in register DE

Decimal code (hex) : 39 (27)

Implementation : All versions of COS and ROS

Registers affected : DE

BASE returns the address of the base of a defined area of firmware RAM workspace in register DE. This instruction is provided to cope with variations in workspace addresses in firmware versions.

You can install alternative handlers (see chapter 13) in place of an EMT, if it is called using a transfer vector. To do this, you have to adjust the contents of the transfer vector to point to your own routine.

The absolute address of a transfer vector can differ between each firmware version, but the displacement of that address from the defined workspace base is always the same.

Consequently, chapter 13 contains a table of offset addresses with respect to this base. To find where the defined workspace memory base is in any version, use BASE.

VERSN

Return version number of the firmware

Decimal code (hex) : 51 (33)

Implementation : All versions of COS, (except 3.0) and all versions of ROS.

Registers affected : AF, HL, DE, and BC.

VERSN returns the COS or ROS version number in register A:

- In COS, the value returned is the version number multiplied by 10 to convert it to an

integer. For example, for COS 3.4 a return value of 34 is generated in register A.

- In ROS, the version number multiplied by 10, plus 100, is returned. For example, ROS 1.0 gives 110 in register A.

This instruction is useful when writing transferable software. You can write a program which finds out what system it is running on, then adapts its operation accordingly. The program finds the system by a call to EMT VERSN.

Unfortunately, the instruction did not exist in COS 3.0. If you want to use VERSN and still have a program that runs on COS 3.0 systems, you must modify the TRAPX transfer vector to simulate the action of the EMT on COS 3.0 systems (see chapter 13).

MOVBLK

Copy a block of memory to another area

Decimal code (hex) : 61 (3D)

Implementation : ROS 1.0 only

Registers affected : None

MOVBLK moves a copy of a block of memory from one area of RAM to another. It allows access to RAM outside the normal 64K addressing range. The number of bytes to be moved is passed in register IX (maximum 16Kbytes).

The block is addressed by a high order register (using bits 0 and 1 only) and a normal register pair (using all 16 bits). If the high order register C contains zero, the register pair DE will address the normal 64K of memory.

The data is copied into the area with start address given by high order register B and register pair HL.

If any of the addresses point to RAM which is not present, there is an immediate return from the instruction with no effect. Interrupts are disabled during this instruction. Beware of copying the workspace area of RAM, since this could have catastrophic results.

RAMMAP

Find amount of available memory

Decimal code (hex) : 62 (3E)

Implementation : ROS 1.0

Registers affected : A

RAMMAP is used to find the amount of available memory. On entry, register B must contain 0. On exit, register A will contain the number of 16K blocks of RAM that are available.

RAMMAP modification

Implementation : ROS 1.1, 1.2, and 2.2

As above, but register B can contain 1. In this case, register HL contains a bit map of the available blocks; bit 15 represents block zero, bit 14 represents block 1, and so on up to bit 0 for block 15.

CHAPTER 10

DEBUGGING FACILITY

This chapter gives details of a very useful facility provided by COS/ROS: the Front Panel. This is a display showing:

- the contents of the Z80 registers and the memory that they address
- the contents of the I/O ports at the displayed address
- the addresses and contents of a 32-byte block of memory.

You can modify the registers and memory contents, and there are very useful commands that allow you to test programs for bugs.

Of course, there are many debugging utilities on the market; DDT is an example. However, they have to be loaded into RAM, taking up valuable memory space. The Front Panel has the advantage that it is part of the firmware, so it does not take up any RAM space.

Instructions on how to use the Front Panel for debugging programs are given in the Research Machines manual: Machine Language Programming Guide. This chapter is a reference document covering the modifications that have been made with new firmware implementations. The first section describes the Front Panel, and the second section gives details of Front Panel commands.

THE FRONT PANEL

You can enter the Front Panel from COS/ROS command level by entering <CTRL/F>. To enter during the execution of a program, you can do one of the following three things:

- Use 0FF hex, the breakpoint code.
- Enter <CTRL/F> when the program is waiting for input from the keyboard.
- Call an entry address
- In ROS versions, type <CTRL/SHIFT/9>.

In each case (except the first one), you can return control to the program by using the K command. Control returns to the line following the line executed last (assuming that the contents of the program counter have not been changed).

The Front Panel Display

When you enter the Front Panel, the display on the screen will look similar to Figure 10.1.

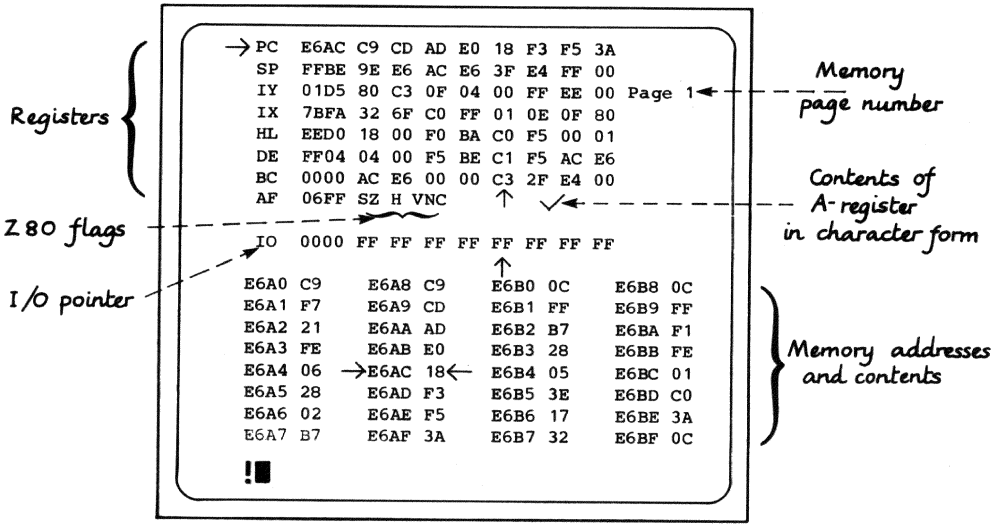


Figure 10.1 The Front Panel Display

In firmware versions earlier than ROS 1.1, there are some differences from this display:

- ROS 1.0 and all COS versions do not display the memory map page number.
- COS 3.0, 3.4 and 4.0 do not display the accumulator contents in character form.

The Front Panel prompt for a command is the ! sign followed by the cursor. Only the bottom four lines of the screen are scrolled; the top 20 lines are used for displaying three zones:

1. The register display zone (the upper 8 displayed lines)
2. The I/O port zone (the middle line)
3. The memory display zone (the lower 8 displayed lines)

The upper zone of the Front Panel display shows the current contents of the Z80 registers and of the memory locations that they address. The display relating to each register occupies a single row and registers are

identified using their standard Zilog abbreviations. From the top of the screen downwards these are as follows:

- The program counter (PC)
- The stack pointer (SP)
- The index registers (IX and IY)
- The 16-bit general-purpose register pairs (HL, DE and BC)
- The accumulator and flag register (AF)

When the alternative set of registers is displayed, their names are tagged with apostrophes: HL', DE', BC', AF'.

On the lefthand edge of the register area, an arrow indicates the register or register pair currently selected for modification. This arrow points initially to the program counter.

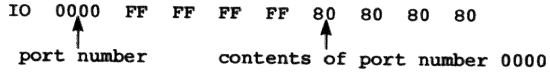
Immediately to the right of each register name, its contents are displayed as a four-digit hexadecimal number; to the right of this is shown the contents of each of the eight bytes in the memory region addressed by the register.

The byte currently addressed by the register pair is in the column with an upwards pointing arrow at its foot. To the left, in a given row, are the contents of locations with lower addresses than the current register content, and to the right are the contents of locations with higher addresses.

For the PC, SP, IY, IX, HL, DE, and BC registers, the four bytes preceding the current location, and the three bytes following it, are displayed.

In the case of the accumulator, the contents of the flag register, F, are displayed when set. The standard Zilog flag abbreviations are used: S, Z, H, V, N, and C. On the righthand side of this line, the present contents of the accumulator are displayed in character form.

The row between the register and memory displays shows the contents of I/O ports:



The 4-digit number indicates the port number, and the vertical arrow points at the contents of that I/O port. In the above example, the contents of port0 are 80 hex.

The three 2-digit numbers on the right of the arrowed contents contain the contents of ports 0001, 0002, and 0003 (extreme right). The four 2-digit numbers on the left contain the contents of ports FFFF, FFFE, FFFD, and FFFC (extreme left).

The lower zone of the Front Panel display shows the address and contents of a 32-byte area of memory that is centred on the current memory address. This is the address pointed at by the two arrows on the fifth row of the second column; initially it is 0100. In figure 10.1, the arrows point at 0E6AC.

FRONT PANEL COMMANDS

Front Panel commands are entered through the keyboard. The majority of them are single letter commands. Figure 10.2 summarizes them.

Figure 10.2 Front Panel Commands

Definitions of the commands are handled in seven groups: pointer commands, memory and register modification, input/output port commands, jump and step commands, search and calculate commands, the outside world, and latest commands.

Pointer Commands

Table 10.1 gives the family of commands that is used to set the memory pointer and to move the register pointer.

Table 10.1 Front Panel pointer commands

Command	Function
<RETURN>	Move memory address forward one byte
<LINE FEED>	Move memory address forward 8 bytes
<CTRL/L>	Move memory address forward 32 bytes
<-> (minus)	Move memory address back one byte
</> (slash)	Move memory address back 8 bytes
<CTRL/O>	Move memory address back 32 bytes
<.> (period)	Move register pointer/set the register address

Table 10.2 describes firmware implementation and differences.

Table 10.2 Implementation and differences

Command	COS 3.0	COS 3.4	COS 4.0	COS 4.2	ROS 1.0	ROS 1.1	ROS 1.2	ROS 2.2
<RETURN>	*	●	*	*	*	*	*	*
<LINE FEED>	*	●	*	*	*	*	*	*
<CTRL/L>	*	●	*	*	*	*	*	*
<->	*	●	*	*	*	*	*	*
</>	*	●	*	*	*	*	*	*
<CTRL/O>	*	●	*	*	*	*	*	*
<.>	*	●	*	*	*	*	*	*

● Implemented
* Identical

Two arrows, known as the memory pointer, point at the fifth line of the second column of memory addresses on the screen display. The address pointed at by the memory pointer is the current address and this can be changed; however, the memory pointer cannot be moved. Instead, the six memory address setting commands move the whole display of memory addresses with respect to the memory pointer's position.

The one register pointer command, (.), does move the register pointer to any of the eight register pairs or the I/O port.

These commands can also be used to change the contents of a register pair, an I/O port address, or a memory address. By entering a hexadecimal number (up to 4 bytes) before you type the command, the contents at the pointer will be changed, then the move will occur.

For example, to change the contents at the present memory address to 0FF hex, and to move the memory address forward 8 bytes, enter:

FF <LINE FEED>

Definitions of the commands are listed below:

RETURN

Move memory address forward 1 byte

Implementation : All versions of COS and ROS
This command advances the memory address by one location with respect to the current address.

LINE FEED

Move memory address forward 8 bytes

Implementation : All versions of COS and ROS
This command advances the memory address by 8 locations with respect to the current address.

CTRL/L

Move memory address forward 32 bytes

Implementation : All versions of COS and ROS

This command advances the memory address by 32 locations with respect to the current address.

- (minus)

Move memory address back by one byte

Implementation : All versions of COS and ROS

This command moves the memory address backwards one location with respect to the current address.

/ (slash)

Move memory address back 8 bytes

Implementation : All versions of COS and ROS

This command moves the memory address backwards by 8 locations with respect to the current address.

CTRL/O

Move memory address back 32 bytes

Implementation : All versions of COS and ROS

This command moves the memory address backwards by 32 locations with respect to the current address.

. (period)

Move register pointer/set the register pair

Implementation : All versions of COS and ROS

This command moves the register pointer down one position to point at the next register or at the I/O port. If the pointer is pointing at the I/O port when you enter <.>, it jumps upwards to point at the PC register.

To set the register pair, enter a 4-digit hexadecimal number followed by <.>. For example, to change the contents of register pair BC from 0012 to 3412, move the pointer to the BC register and enter:

```
3412 <.>
```

If you entered only 34 <.>, the resultant contents of register BC would be 0034.

Memory and Register Modification

Table 10.3 gives the family of commands that is used for memory and register modification.

Table 10.3 Memory and register modification

Command	Function
<I>	Set memory address from memory contents (abs. indirect word)
<M>	Set memory pointer
<P>	Fill memory
<R>	Set memory address from memory contents (rel. indirect byte)
<S>	Move block of memory
<U>	Update current memory address from program counter
<V>	Update program counter from current memory address
<X>	Exchange register sets

Table 10.4 describes firmware implementation and differences.

Table 10.4 Implementation and differences

COMMAND	COS 3.0	COS 3.4	COS 4.0	COS 4.2	ROS 1.0	ROS 1.1	ROS 1.2	ROS 2.2
<I>	*	●	*	*	*	*	*	*
<M>	*	●	+	+	+	+	+	+
<P>	*	●	+	+	+	+	+	+
<R>	*	●	*	*	*	*	*	*
<S>	*	●	+	+	+	+	+	+
<U>	*	●	*	*	*	*	*	*
<V>	*	●	*	*	*	*	*	*
<X>	*	●	*	*	*	*	*	*

● Implemented
* Identical
\+ Almost the same

This group of commands is made up of three types:

- those that set the memory address (I, M, R, U)
- those that influence the memory contents (P, S)
- those that modify the registers (V, X).

Definitions of the commands are listed below:

I

Set memory address from memory contents (absolute word, indirect)

Implementation : All versions of COS and ROS

This command sets the memory address to the 16-bit address which is held in the current location of the memory pointer.

The command expects the least significant byte first, as in Z80 addressing.

M

Set memory pointer

Implementation : COS 3.0 and 3.4

This command sets the memory pointer to any location. The entry prompt is displayed when you press <M>; enter a hexadecimal number (up to 4 digits) followed by <RETURN>. This now becomes the memory address pointed to by the memory pointer.

M modification

Implementation : COS 4.0 and 4.2.
All versions of ROS

The entered hexadecimal number can only be terminated by <RETURN>, <SPACE>, or <CTRL/F> (which aborts the command). All non-hexadecimal characters, other than <DELT>, are ignored.

P

Fill memory

Implementation : COS 3.0 and 3.4

This command fills a section of memory with one particular byte of your choice. There are prompts for the first and last addresses of the area to be filled, and for the byte to fill it. If the last address is lower than the first, an error message is displayed and control is returned to the COS command level.

While memory is being filled, the system reads it back to make sure that the new contents have been stored correctly, thus providing a simple check of the memory. If a location does not read back correctly, an error message is displayed and the Front Panel display is updated

so that the memory pointer indicates the location at which the error has been detected.

P modification

Implementation : COS 4.0 and 4.2.
All versions of ROS

Entered hexadecimal numbers can only be terminated by <RETURN>, <SPACE>, or <CTRL/F> (which aborts the command). All non-hexadecimal characters entries, other than <DELT>, are ignored.

The sequence is restarted if the last address is greater than the first.

R

Set memory address from memory contents (relative byte, indirect)

Implementation : All versions of COS and ROS

This command sets the address pointed to by the memory pointer to the relative address, calculated from the present memory contents.

The value of these contents is taken as the number of locations that the present memory address must be moved. An extra one must be added to this expected relative jump, to account for the automatic advance of the program counter. The command sets the pointer to the address that the Z80A would reach on encountering this relative jump.

S

Move block of memory

Implementation : COS 3.0 and 3.4

This command shifts the contents of a portion of memory into another portion of memory.

Enter <S>, and prompts are displayed requesting the first and last address of the block of memory to be moved, and then for the first address of the area into which the block is to be moved. All three addresses must be entered as hexadecimal values up to 4 digits in length.

A block of any size can be moved in any direction, but if the first address of the block to be moved is higher than the second, an error message is displayed and control is

returned to COS command level.

S modification

Implementation : COS 4.0 and 4.2.
All versions of ROS

In the case of erroneous entry, an error message is displayed but control is not transferred to COS/ROS command level and the sequence is restarted.

The hexadecimal numbers may be terminated by <RETURN>, <SPACE>, and <CTRL/F> (which aborts the command). All non-hexadecimal characters, other than <DELT>, are ignored.

U

Update memory address from program counter

Implementation : All versions of COS and ROS

This command sets the memory pointer to the present contents of the program counter.

V

Update program counter from current memory address

Implementation : All versions of COS and ROS

This command sets the program counter contents to the current memory pointer.

X

Exchange register sets

Implementation : All versions of COS and ROS

This command displays the alternative set of registers (HL', DE' BC' and AF'). All operations on registers now affect this alternative set until <X> is pressed again.

Input/Output Port Commands

These commands are used to enter values into the current output port, and to change the current I/O port address. Table 10.5 gives a summary of these commands and Table 10.6 describes firmware implementation and differences.

Table 10.5 Input/output port commands

Command	Function
<,> (comma)	Write to I/O port
<:> (colon)	Set I/O port number
<<> (less than)	Increment port address
<>> (greater than)	Decrement port address

Table 10.6 Implementation and differences

COMMAND	COS 3.0	COS 3.4	COS 4.0	COS 4.2	ROS 1.0	ROS 1.1	ROS 1.2	ROS 2.2
<,> (comma)	*	●	*	+	*	+	+	+
<:> (colon)	X	X	X	●	*	*	*	*
<<> (less than)	*	●	*	*	*	*	*	*
<>> (greater than)	*	●	*	*	*	*	*	*

● Implemented + Almost the same
* Identical X Not implemented

Command definitions are listed below:

, (comma) Write to I/O port

Implementation : COS 3.0

This command inserts a new value into the current output port specified by the vertical pointer. The destination of this value is controlled by the port address (see appendix C for I/O port allocations).

If no hexadecimal value is entered before you enter <,>, zero is sent to the output port.

, (comma) modification

Implementation : COS 3.4 and 4.0, ROS 1.0

If a comma is entered without any hexadecimal characters, the command is ignored.

Implementation : COS 4.2
ROS 1.1, 1.2 and 2.2

The ports displayed are not read back as part of the Front Panel response to the command.

: (colon)

Set I/O port number

Implementation : COS 4.2 and all versions of ROS

This command changes the current I/O port address without the necessity of positioning the register pointer on the I/O line of display.

To change the address, enter your required new address (up to 4 hexadecimal digits) followed by <:>. The I/O port address will immediately be changed to this new address.

< (less than)

Increment port address

Implementation : All versions of COS and ROS

This command increases the I/O port address by one location. The displayed row of port contents are shifted one place to the left.

> (greater than)

Decrement port address

Implementation : All versions of COS and ROS

This command decreases the I/O port address by one location. The displayed row of port contents are shifted one place to the right.

Jumps and Steps

These commands are useful for testing programs that you have written. It is possible to execute one instruction (Z command), or one sub-routine (Y command), at a time. After execution, the register and memory displays are updated to reflect any changes that have occurred. Table 10.7 summarizes the jump and step commands; table 10.8 lists implementation and version difference details.

Table 10.7 Jump and step commands

COMMAND	FUNCTION
<J>	Jump to address and start program execution
<K>	Continue program execution from PC address
<Y>	Step through calls, EMTs, and repeated instructions such as LDIR
<Z>	Single step through individual program instructions

Table 10.8 Implementation and differences

COMMAND	COS 3.0	COS 3.4	COS 4.0	COS 4.2	ROS 1.0	ROS 1.1	ROS 1.2	ROS 2.2
<J>	*	●	+	+	+	+	+	+
<K>	*	●	+	+	+	+	+	+
<Y>	X	X	X	X	X	●	*	*
<Z>	*	●	+	+	+	-	-	-

●	Implemented	+	Almost the same
*	Identical	-	Major differences
		X	Not implemented

J

Jump to address and start program execution

Implementation : COS 3.0 and 3.4

This command can be used to start programs without setting the program counter to the start address.

When you enter <J>, the entry prompt is displayed. Enter the required address and press any key. If you enter the wrong address, it can be cancelled by entering <CTRL/F>.

J modifications

Implementation : COS 4.0 and 4.2, ROS 1.0

After entering your address, the only terminating characters allowed are <RETURN>, <SPACE> or <CTRL/F> (to cancel). All other non-hexadecimal characters (except <DELT>) are ignored.

The screen is cleared with reversion to its entry screen width.

Implementation : ROS 1.1, 1.2 and 2.2

If saved, the screen is restored. Otherwise the screen is cleared.

K

Continue program execution from PC address

Implementation : COS 3.0 and 3.4

This command restores all registers and continues execution at the address currently in the program counter. The screen is not cleared.

K modifications

Implementation : COS 4.0 and 4.2, ROS 1.0.

When the <K> command is entered, the screen is cleared with reversion to the entry screen width.

Implementation : ROS 1.1, 1.2 and 2.2

If saved, the screen reverts to the saved image. Otherwise the screen is cleared.

Y

Step through calls and EMTs

Implementation : ROS 1.1, 1.2, and 2.2

This command allows you to execute a subroutine as though it is a single instruction.

If execution could involve screen display, the Front Panel display reverts to the saved image while the sub-routine is being executed, then displayed again at the end of execution.

Z

Single step through individual program instructions

Implementation : COS 3.0 and 3.4

This command allows you to execute a program one instruction at a time.

Each time <Z> is pressed, one instruction is executed; the register and memory displays are updated if changes have occurred.

If execution involves screen display, this is superimposed on the Front Panel display.

Z modification

Implementation : COS 4.0 and 4.2

This command executes an entire EMT instruction rather than single-stepping through it.

Implementation : ROS 1.0

This command executes an entire CALR instruction (see chapter 14) as well.

Implementation : ROS 1.1, 1.2, and 2.2

If execution could involve screen display the Front Panel display reverts to the saved image while the instruction is executed, then displayed again at the end of execution.

Search and Calculate Commands

The G and N commands enable you to search the memory for a specified pattern of bytes. The @ and H commands are calculators that assist in working out relative addresses.

Table 10.9 summarizes these commands; Table 10.10 describes firmware implementation and differences.

Table 10.9 Search and calculate commands

COMMAND	FUNCTION
<@>	Calculate and insert relative offset
<G>	Search for specified pattern of bytes
<H>	Hexadecimal calculator
<N>	Find next occurrence of a pattern

Table 10.10 Implementation and differences

COMMAND	COS 3.0	COS 3.4	COS 4.0	COS 4.2	ROS 1.0	ROS 1.1	ROS 1.2	ROS 2.2
<@>	X	X	X	X	X	●	*	*
<G>	*	●	+	+	+	+	+	+
<H>	*	●	+	+	+	+	+	+
<N>	*	●	*	*	*	*	*	*

● Implemented
* Identical
\+ Almost the same
X Not implemented

Command definitions are given below:

@

Calculate and insert relative offset

Implementation : ROS 1.1, 1.2 and 2.2

This command calculates and inserts the relative offset value to be used in a relative jump instruction to jump to the required address. The value is inserted at the present memory address.
For example, if the present memory address is 0100 and you want to perform a relative jump to 0115, you can enter the following simple program.

First use the M command to set the memory to point at address 0100:

<M>0100

Now enter the jump relative machine-code instruction, 18 hex at this address:

18<RETURN>

and the contents of address 0100 will contain 18.

To calculate the relative jump value to jump to 0115, enter:

<@>15<RETURN>

and the contents of address 0101 will now contain 13, the calculated value.

To observe the program in operation, do the following:

- Use the M command again to set the memory pointer at 0100.
- Use the V command to set the program counter contents to 0100.
- Use the Z command to execute the program.

The program counter now points at address 0115.

The entered hexadecimal numbers can be terminated by <RETURN>, <SPACE>, and <CTRL/F> (which aborts the command). All other non-

hexadecimal characters except <DELT> are ignored.

G

Search for specified pattern of bytes

Implementation : COS 3.0 and 3.4

This command searches memory for the entered pattern of bytes (two hexadecimal digits at a time). The pattern can be any reasonable length from one byte upwards. For more than two bytes, the pattern is entered in batches of 2 digits separated by <RETURN>. As soon as <RETURN> is pressed without digit entry, the search begins.

The search stops when the pattern is found, and the memory display is then updated. The memory pointer location contains the first byte of the sought pattern.

G modifications

Implementation : COS 4.0 and 4.2.
All versions of ROS

The entered numbers can only be terminated with <RETURN>, <SPACE>, or <CTRL/F> (which aborts the command).

All other non-hexadecimal characters except <DELT> are ignored.

In ROS 1.1, 1.2, and 2.2 the use of "(double quote) text string <RETURN> is allowed within a search string.

H

Hexadecimal calculator

Implementation : COS 3.0 and 3.4

This command prompts for two hexadecimal numbers (up to 4 digits), each terminated by <RETURN>. Their sum and difference are displayed.

H modification

Implementation : COS 4.0 and 4.2
All versions of ROS

The entered numbers are only terminated by <RETURN>, <SPACE>, or <CTRL/F> (which aborts the command).

All other non-hexadecimal characters except

<DELT> are ignored.

N

Find next occurrence of a pattern

Implementation : All versions of COS and ROS

This command is used in conjunction with the G command. Once a search for a pattern has been set up with the G command, the search for further occurrences of the found pattern can be made by entering <N>.

You can start the search at any location by setting the memory pointer to that location before entering <N>.

The Outside World

This family of commands control exit from Front Panel to COS/ROS command level or to CP/M (CTRL/B and CTRL/C). Temporary exit is provided by the ESC command that allows you to see the screen display from which you entered Front Panel. The O and W commands control certain system features.

Table 10.11 summarizes the commands, and Table 10.12 describes firmware implementation and differences.

Table 10.11 The outside world commands

COMMANDS	FUNCTION
<CTRL/B>	Return to COS/ROS command level
<CTRL/C>	Return to COS/ROS command level or CP/M
<ESC>	Display original screen contents, and return
<O>	Set options
<W>	Switch screen width

Table 10.12 Implementation and differences

COMMAND	COS 3.0	COS 3.4	COS 4.0	COS 4.2	ROS 1.0	ROS 1.1	ROS 1.2	ROS 2.2
<CTRL/B>	*	●	+	+	+	+	+	+
<CTRL/C>	*	●	+	+	+	+	+	+
<ESC>	X	X	X	X	X	●	*	*
<O>	*	●	+	+	*	*	*	*
<W>	X	X	●	*	*	*	*	*

● Implemented
* Identical
\+ Almost the same
X Not implemented

Definitions of these commands are listed below:

CTRL/B

Return to COS/ROS command level

Implementation : COS 3.0 and 3.4

This command returns control to COS command level, but the screen is not cleared of the Front Panel display.

CTRL/B modification

Implementation : COS 4.0 and 4.2
All versions of ROS

The screen is now cleared, the screen width is restored, and the screen contents are restored (if saved).

CTRL/C

Return to COS/ROS command level or CP/M

Implementation : COS 3.0 and 3.4

This command returns control to the most recently used command level, but the screen is not cleared of the Front Panel display.

CTRL/C modification

Implementation : COS 4.0 and 4.2
All versions of ROS

Control returns to the most recently used command level. The Front Panel display is cleared from the screen, the screen width is restored, and the screen contents are restored (if saved).

ESC

Display original screen contents; press again to return to Front Panel.

Implementation : ROS 1.1, 1.2 and 2.2

This command displays the screen display from which you entered the Front Panel.

For example, if you entered Front Panel from the ROS command level, the following firmware sign-on message will be displayed:

RML 80 Character LINK 480Z V 1.2 B
Z-NET Firmware Vers 1.1q Address: XX

To start BASIC in ROM type the command R
Please give a command or type H for help

The computer will not respond to further input, except <ESC> again, which causes the Front Panel to be re-displayed.

O

Set options

Implementation : COS 3.0 COS 3.4/C.
All versions of ROS

This command allows you to select cassette speeds or printer options.

O modifications

Implementation : COS 3.4/M+F, COS 4.0 and COS 4.2

The command no longer checks for cassette speed. It asks only for printer type.

W Switch screen width (80-character systems)

Implementation : COS 4.0 and 4.2.
All versions of ROS.

This command changes the character width to be used on exit from Front Panel. It works only in 80-character systems.

When <W> has been entered, the character width to be used next is displayed.

Latest Commands

This family of commands are recent additions to the list that have not been covered in previous sections. Table 10.13 summarizes these commands, and Table 10.14 describes firmware implementation and differences.

Table 10.13 Latest Front Panel commands

COMMAND	FUNCTION
<"> (double quote)	Enter text string
<\> (back slash)	Change memory page number
<L>	Insert breakpoint
<Q>	Remove breakpoint
<T>	Display memory contents as text or hexadecimal contents

Table 10.14 Implementation and differences

COMMAND	COS 3.0	COS 3.4	COS 4.0	COS 4.2	ROS 1.0	ROS 1.1	ROS 1.2	ROS 2.2
<">	X	X	X	X	●	*	*	*
<\>	X	X	X	X	X	●	*	*
<L>	X	X	X	●	X	*	*	*
<Q>	X	X	X	●	X	*	*	*
<T>	X	X	X	●	*	*	*	*

● Implemented X Not implemented
* Identical

Definitions of these commands are listed below:

" (double quote)

Enter text string

Implementation : All versions of ROS

This command allows you to enter a string of characters terminated by <RETURN>. The text is put into memory sequentially, and the memory pointer points to the location immediately following the last byte inserted.

Entering <CTRL/F> aborts the command, and <DELT> functions normally. Characters with ASCII codes of less than 20 hex are ignored.

This command can also be used in conjunction with the G and N commands to search for a string of characters. Enter <G>, then <"> followed by the character string.

\ (back slash)

Change memory page number

Implementation : ROS 1.1, 1.2 and 2.2

This command changes the currently displayed memory map page to the next is sequence (0, 1, 2, and 4). See chapter 12 for details of memory maps.

L

Insert breakpoint

Implementation : COS 4.2 and ROS 1.1, 1.2, and 2.2

This command inserts 0FF hex (a breakpoint) into the memory contents at the current memory address. This allows you to run a program to a specific point: the breakpoint.

After inserting the breakpoint, you can run your program (using the K or J commands).

When the breakpoint is encountered, the Front Panel is displayed with the state that was present before the breakpoint execution, and the message, BREAK, is displayed.

The program counter and memory pointer now contain the address of the breakpoint. If a new breakpoint is inserted when an old breakpoint is

still present, the old breakpoint is replaced with the original value of the memory contents.

Q

Remove breakpoint

Implementation : COS 4.2 and ROS 1.1, 1.2, and 2.2

This command removes the last breakpoint that was inserted using the L command, restoring the memory content to what it was before the command was initiated.

T

Display memory as text or hexadecimal contents

Implementation : COS 4.2 and all versions of ROS

This command changes the displayed memory contents from their hexadecimal representation to the equivalent characters:

- special graphics (00 to 1F hex, and 7F hex)
- ASCII (20 to 7E hex)
- teletext graphics (80 to 0FF hex).

If you press <T> again, the hexadecimal contents are displayed.

CHAPTER 11

DIRECT ACCESS TO SCREEN MEMORY

This chapter gives details of how to access the screen memory.

As was pointed out in chapter 3, the screen is refreshed at regular intervals from an area of RAM in both 380Z and 480Z machines; but the method of access in each machine is different. Matters are further complicated in COS 4.0 and COS 4.2 380Z machines because there are different accessing method for 40 and 80-character display modes.

Unless it is absolutely necessary, you are strongly advised not to access the screen memory by the methods outlined below. Whenever possible use the EMTs outlined in chapters 2 and 3.

The first section of this chapter describes direct access in 380Z machines; the second section concerns 480Z machines.

380Z MACHINES

COS 3.0 and 3.4

The screen display is a direct display of a 960-character block of static memory, arranged as 24 lines of 40 characters each; this block is at locations F000 to F5FF (see chapter 12). It is accessible to both video circuitry and the CPU, but not simultaneously.

Control of access to the screen memory is provided by the state of bit 2 of the memory-mapped port, PORT0 (see chapter 12):

- 0 grants access to the video
- 1 grants access to the VDU

Consequently, you could access the screen memory with your program at any time by setting the control bit in PORT0. But, if this occurs when information is being sent to the screen, the picture is disturbed. There will be a bright flash, or a momentary dark horizontal band.

To prevent these disturbances, you can synchronize your program with the video circuitry. The CPU can take control only during periods when information is not being sent to the screen; those periods are during frame and line blanking. Two bits in PORT1 (see chapter 12) indicate when they occur:

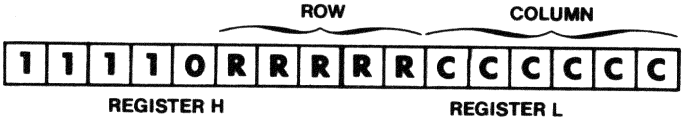
- Bit 6 is set during the frame blanking period. This is approximately 4.5 milliseconds every 20 milliseconds.

- Bit 7 is set during the line blanking period. There is time to output only one character.

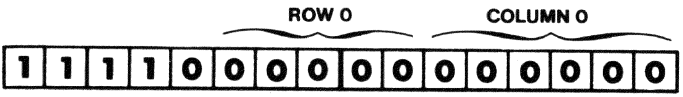
The EMT instructions OPNWT and CLOSE (see chapter 4) simplify matters further. OPNWT waits until the next frame blanking period occurs, then it opens the screen memory for program access. CLOSE closes the screen memory to the CPU.

Provided that instructions take less than 4.5 milliseconds to execute, your code to manipulate screen memory can be placed between calls to OPNWT and CLOSE without disturbing the current picture display.

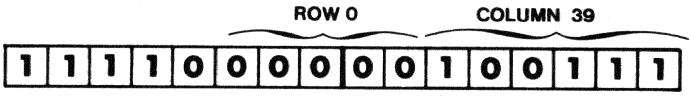
Characters can be sent to any part of the screen using a 16-bit number in register pair HL. This number gives the row address (R) and column address (C) in the form:



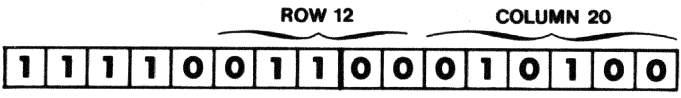
The top lefthand corner of the screen (row 0, column 0) is given by:



The top righthand corner (row 0, column 39) is given by:



A central screen position (for example row 12, column 20) is given by:



In short:

- choose your row number and column number
- convert the respective numbers into binary code
- enter the binary number in the correct area of the 16-bit number which makes up the address in register pair HL.

The following program example shows how to display the letter A near the top lefthand corner of the screen, third row down, tenth character along:

```
ld      hl,3*64+10+0f000h    ;Load HL with value 0f000H
                             ;(top lefthand corner) + offset
emt     opnwt                ;Wait for screen blanking.
ld      (hl),'A'             ;Load character A to
                             ;screen.
emt     close                ;Close screen.
```

COS 4.0 and 4.2

These firmware versions support both the 40-character and 80-character modes of screen display. Direct access to the screen memory is different for the two modes:

- 40-character mode

  As described in the previous section.

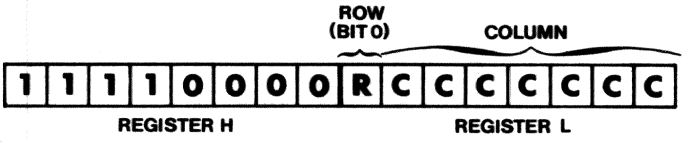
- 80-character mode

  The screen memory is mapped differently, and not all the screen can be addressed at the same time.

  The character position addressing procedure is different. Choose your row and column numbers, and convert them to binary code.

  For the column address, the binary number is placed in the righthand bits of the 16-bit number (as in 40-character mode) . As the number of addressable columns is now 80, one more bit is needed for numbers greater than 63.

  For the row address, only bit 0 of the binary row number is put in the HL register pair. The remaining bits 1 to 4 are put in bits 0 to 3 of PORT 1. Consequently, the HL register pair will contain:



Here is how to display the letter in the same place as above:

```
ld      hl, 128&(3*64)+10+0f000h
ld      a,(mask1)
and     0F0h
or      a, 3 shr 1              ;Shift row right 1 bit.

                                ;Register A now has
                                ;the required value for
                                ;PORT1.

ld      (port1),a
emt     opnwt                   ;As above.
ld      (hl),'A'
emt     close
```

480Z MACHINES

The 480Z does not have any memory-mapped ports. Input/output-mapped ports (see appendix C) are accessed by Z80 I/O instructions.

In both 40 and 80-character modes the 480Z screen memory is mapped as a block of 24 input/output ports that the CPU can access at any time. the port at I/O address 0 corresponds to the top line of the screen, and the port at 17 hex (23 decimal) corresponds to the bottom line.

Any of the Z80 I/O instructions using register C (such as IN, OUT, INIR, OTIR), that are described in any Z80 programming manual, can be used to read from or write to the contents of the screen memory in all versions of ROS. The row number should be placed in register C and the column number in register B before entering the instruction.

For example, to display the letter Z at the bottom righthand corner of the screen (line 23, column 80), the following instructions could be used:

```
ld      b,80
ld      c,23
ld      a,'Z'
out     (c),a
```

A whole line (80 characters) of text can be read from the screen to RAM by a single INIR instruction, or written out with an OTIR instruction.

Note that there is no need to open or close the screen to RAM before or after access, as you do in COS. But, the line blanking and frame blanking signals are accessible as bits in a port; these signals can be used for timing purposes.

CHAPTER 12

MEMORY LAYOUT

When operating in a high-level language, the computer allocates memory space to programs and variables in a manner determined by the language system designer. Though you can overide this automatic memory allocation, it is rare that you need to do so.

In assembly language, however, you often have to make decisions about memory allocation. Consequently, you must know the layout of both RAM (random access memory) and ROM (read only memory). You need to know where the operating systems (both firmware and CP/M) are stored, and which areas of memory you can use.

This chapter has sections on usable memory, reserved memory, and memory pages.

USABLE MEMORY

The Z80 microprocessor can address 64Kbytes of memory. Each location is referred to (in hexadecimal notation) by its position in the memory. The first location, at the bottom of the memory, is 0000H, the next is 0001H, the next is 0002H, and so on up to 0FFFFH.

The normal layout of memory, with the CP/M operating system loaded, is shown in figure 12.1.

The total size of memory available depends on your computer system:

- Very early 380Z machines had a variety of memory sizes up to 32Kbytes.
- The latest 380Z machines have 64Kbytes.
- Some early 480Z machines have 32K; later models have 64K.

With CP/M loaded, the available RAM (in the transient program area) lies between 0100H and the address in location 0006H (HIMEM); this points to the top of usable RAM plus 1. On 64Kbyte machines this gives approximately 50K of usable memory. Machines with smaller memory size have less usable RAM (smaller transient program area).

The amount of usable RAM can be increased by not loading CP/M, or by overwriting the loaded CP/M utilities in memory; this increases usable memory by about 8K, but you have only the firmware facilities. Disc-handling operations are curtailed by this action.

It is strongly recommended that, whenever possible, you write your programs using CP/M facilities; this assures maximum transferability of your software. For most programs, there is sufficient usable memory, even with CP/M loaded.

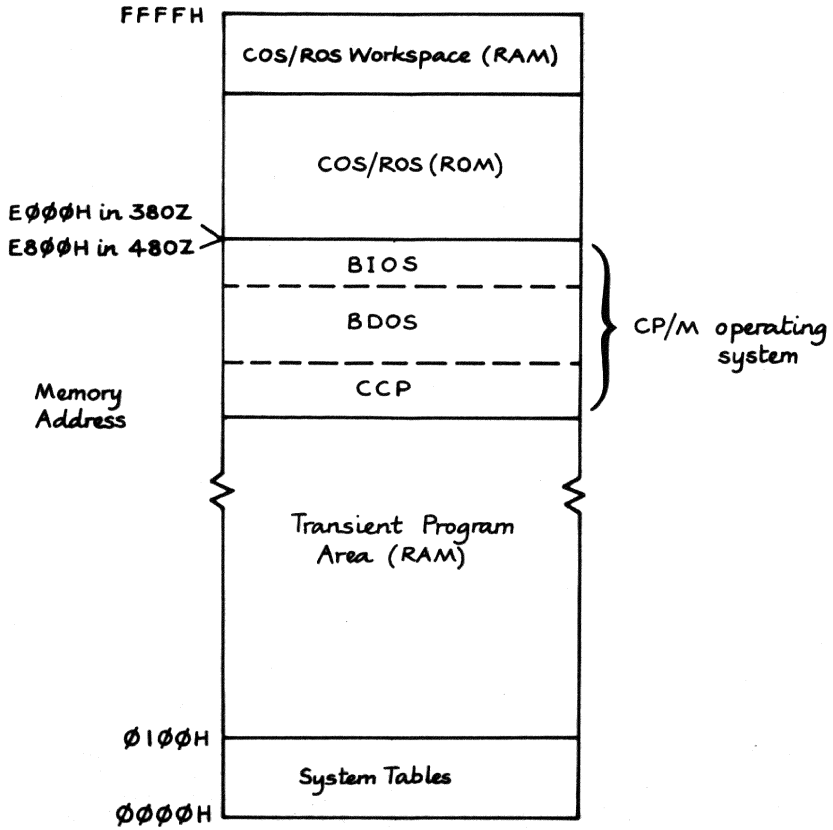


Figure 12.1 Memory layout with CP/M loaded

RESERVED MEMORY

With a few exceptions, the areas of memory referred to in this section are not accessible to you. The layout details given here are for information only and the contents of these locations should not be tampered with under any circumstances.

Figure 12.2 shows the layout of firmware facilities at the top of memory (above E000H in figure 12.1). The component parts are discussed below.

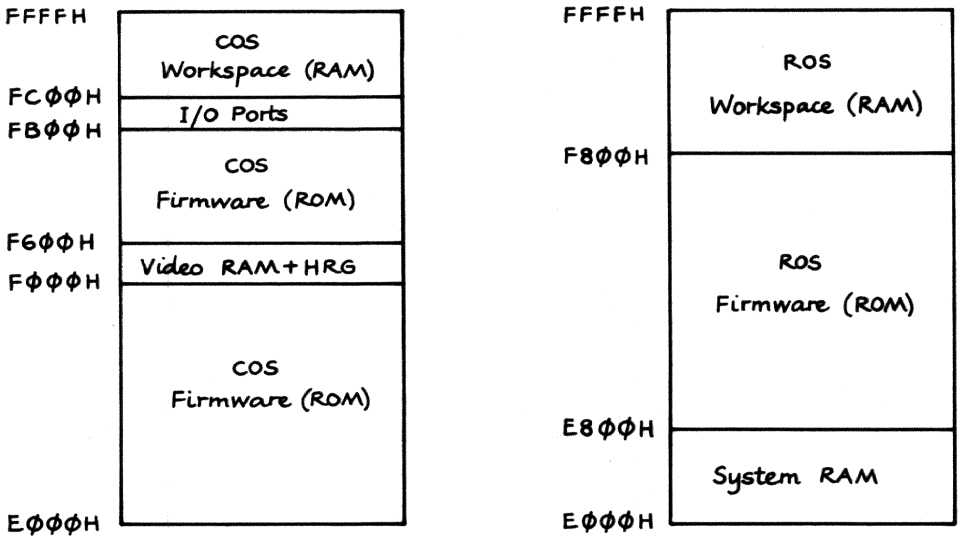


Figure 12.2 Layout of firmware memory usage above E000H

COS Firmware (ROM)

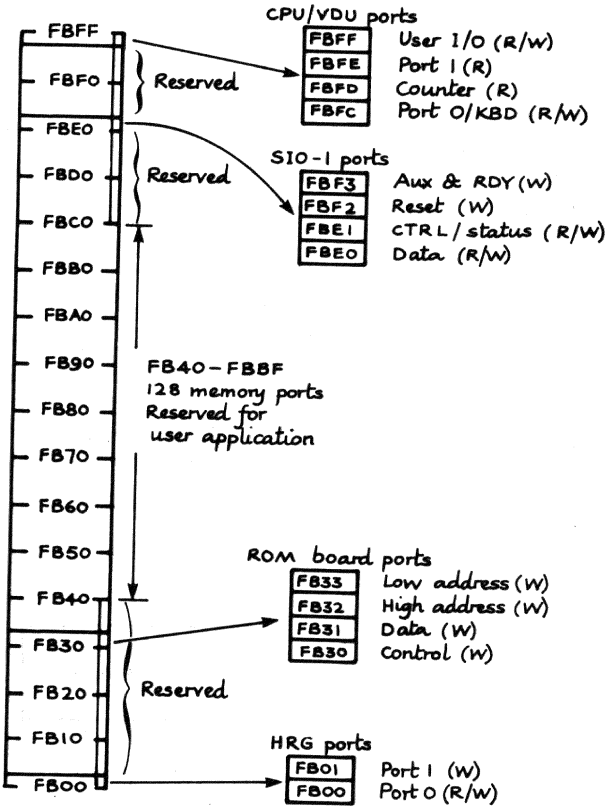
Areas 0E000H to 0FFFFH and 0F6000 to 0FAFFH in figure 12.2(a) should not be used as they change with every version.

COS Video RAM +HRG

Area 0F000H to 0F5FFH in figure 12.2(a) is used for screen display by direct access to memory (see chapter 11).

I/O Ports in a 380Z

Area 0FB00H to 0FBFFH in figure 12.2(a) is the 380Z I/O memory port area. Details of the ports are shown below:



Bit allocations in PORT0 and PORT1 are given in table 12.1

Table 12.1 Bit allocation in PORT0 and PORT1(CPU/VDU ports)

Bit	PORT0 (write)	PORT1 (read)
0	clear keyboard latch	-
1	enable single step (0=set)	1200 Hz clock
2	open screen memory	reset button
3	relay 1 (0=closed)	cassette volume sense
4	set 2400 Hz	-
5	relay 2 (0=closed)	cassette signal sense
6	clear 8 microsecond counter	screen frame waveform
7	memory page switch	screen line waveform

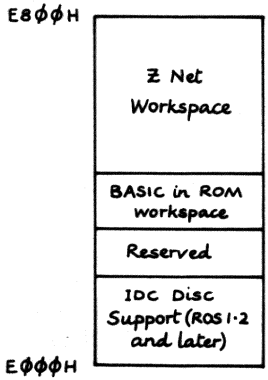
The 380Z uses I/O-mapped ports as well; details of these are given in appendix C.

ROS Firmware (ROM).

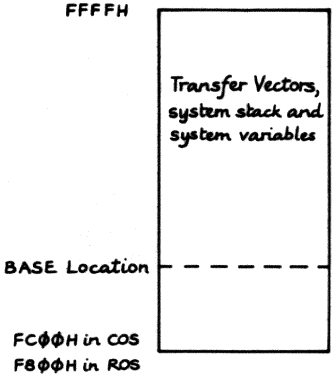
Area 0E800H to 0F800H in figure 12.2(b) should not be used as it changes with every version.

ROS System RAM

Area 0E000H to 0E800H in figure 12.2(b) is the ROS system RAM area. This is out of bounds and, for information only, its contents are shown below:



COS And ROS Workspace (RAM)



This area of RAM, known as firmware workspace, can only be accessed for the addition of device handlers (as outlined in the next chapter) or to use MASK. Otherwise, do not use this area; there is a big danger that you will damage the system.

System Tables

Area 0000H to 0100H in figure 12.1 is the system tables area. It is split into two parts:

- 0000H to 007FH, an area of RAM that is used by both firmware and CP/M (see the CP/M Programmers Manual for a description of how CP/M uses this area).
- 0080H to 00FFH, an area of RAM that can be used with caution.

The firmware use of RAM area is shown in figure 12.3, and the CP/M use of this same area is shown in figure 12.4. Figure 12.5 illustrates the areas that overlap and sometimes cause confusion.

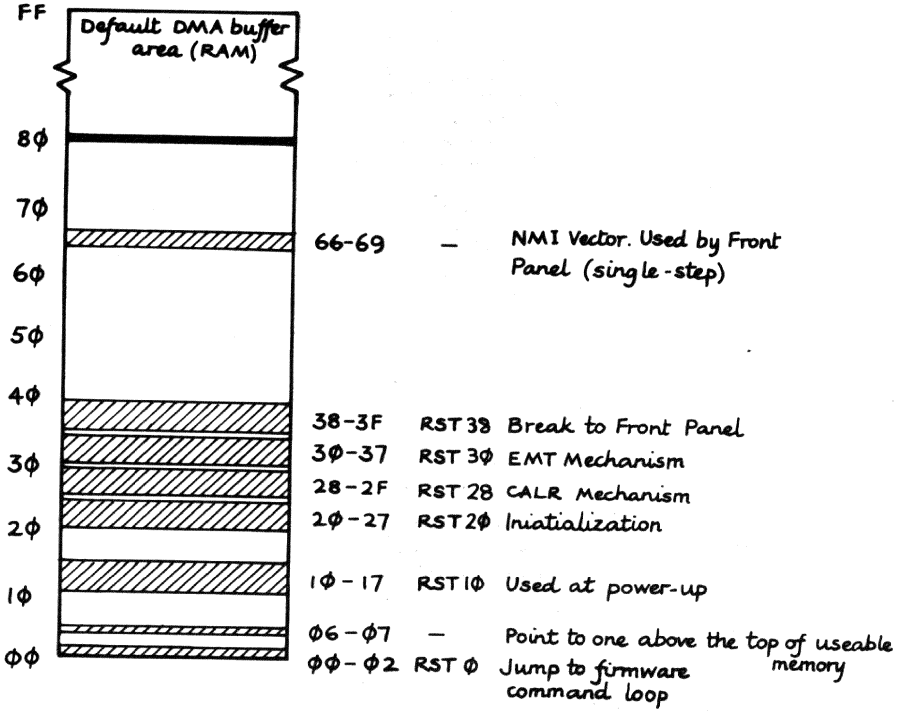


Figure 12.3 Firmware use of system tables area

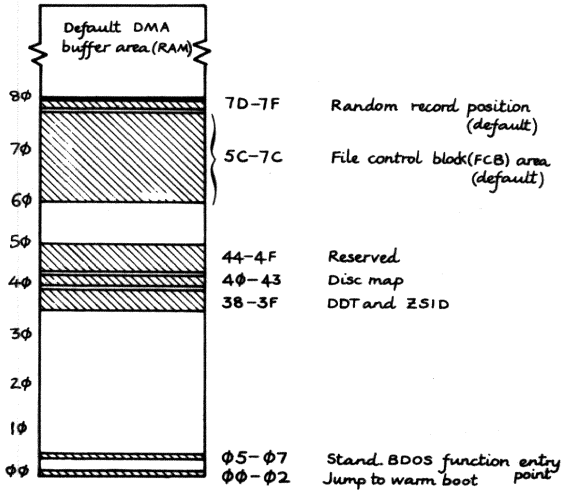


Figure 12.4 CP/M use of system tables area

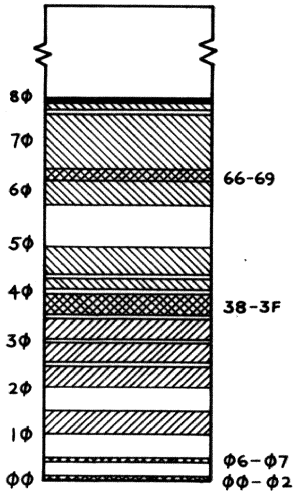


Figure 12.5 Overlapping use of system tables area

MEMORY PAGES

With the 480Z, it is possible to have more than one memory layout. There are four layouts with different addressing schemes, or pages. These pages consist of blocks of ROM and RAM; the mapping of ROM into address space is controlled by a mapping PROM, and the mapping of RAM is controlled by the firmware.

The layout shown in figure 12.1, known as page 1, is most commonly used. The other pages are:

- page 0 Initialization, some ROS functions
- page 2 BASIC in ROM
- page 3 ROS (not accessible)

Interrupt Routines

Since paging exists, all interrupt routines should be in an area of memory that is present in all pages. Such an area is the top 16K of memory. The address of the interrupt vector table is FC00H. For further information, see the Mostek Z80 Programming Manual. For use of vectors in the table, see the 480Z Information File.

CHAPTER 13

TRANSFER VECTORS AND DEVICE HANDLERS

This chapter gives details of how the EMT mechanism works. The differences in operation between directly-called EMTs and EMTs called using transfer vectors are highlighted. The addition of your own routines, or device handlers, is discussed.

The first section of the chapter explains how the EMT mechanism operates. The second section concerns transfer vectors, device handlers, the TRAPX vector, and filters.

HOW THE EMT MECHANISM WORKS

The actions involved in an EMT call (and the areas of memory that are used) are outlined in figure 13.1, using graphical representations of the page 1 memory layout.

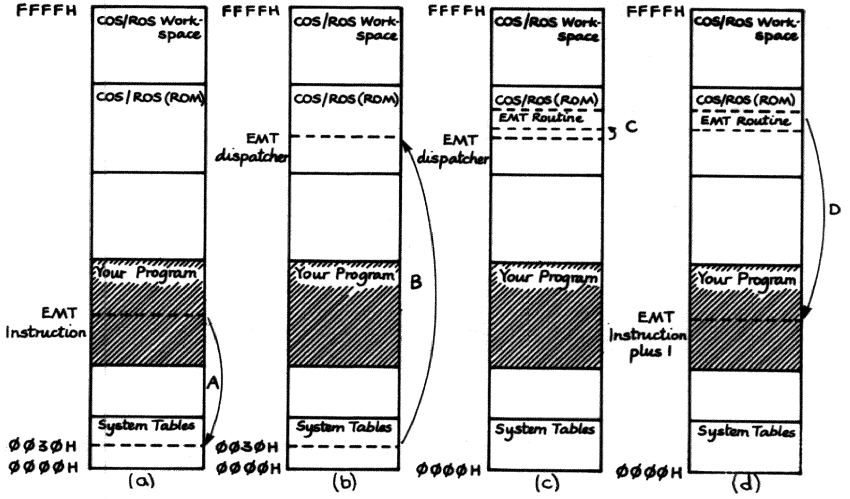


Figure 13.1 How the EMT mechanism works

Your program requests execution of an EMT instruction using the RST 30H instruction. This produces jump A (figure 13.1(a)) to location 30H in the system tables area of memory.

The contents of location 30H produce a jump to the EMT dispatcher in the COS/ROS (ROM) area, jump B in figure 13.1(b).

The code number of the required EMT (in the byte following the RST 30H command) is evaluated, and a jump to the start of the EMT routine occurs (jump C in figure 13.1(c)). When the routine has been executed, control returns to your program at the instruction following RST 30H (jump D in figure 13.1(d)).

The simple program example on page 1.6 shows how the mechanism is used in practice.

TRANSFER VECTORS

In most cases, the mechanism described above provides ample scope for programming requirements. However, some EMT instructions are called using a 3-byte transfer vector. In this case, jumps A and B occur as before, then the mechanism changes as shown in figure 13.2.

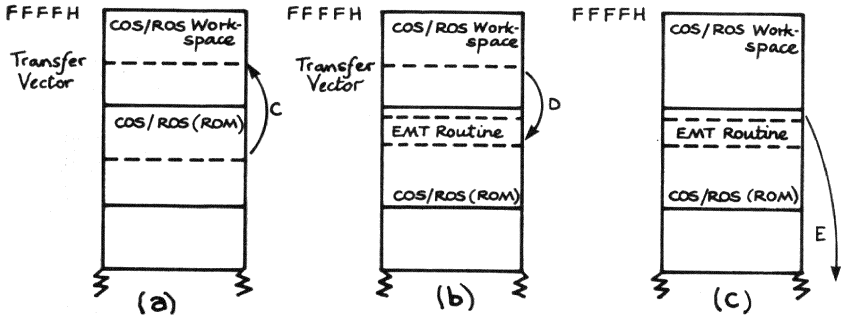


Figure 13.2 The EMT mechanism using a tansfer vector

The EMT dispatcher checks that the EMT is called by a transfer vector, and a jump to the relevant transfer vector takes place (jump C in figure 13.2(a)). The transfer vector contains a jump instruction in the first byte, and the address of the start of the EMT routine (in bytes 2 and 3).

Consequently, jump D (in figure 13.2(b)) occurs and, when execution of the routine is completed, command returns to your program (jump E in figure 13.2(c)).

Device Handlers

You can intercept the EMT mechanism and add your own routines by replacing the address in the transfer vector with the start address of your own routine (or device handler). In this way, jump D would go to your routine situated in another part of memory, rather than to the EMT routine.

To be able to do this, you need to know the location of the transfer vector. The absolute address of a transfer vector is different in each firmware version, but the displacement of the address from the base of a defined area of workspace is always the same. You can find this displacement for all transfer vectors in table 13.1; this gives offset addresses from base.

In ROS, your device handlers must reside in the top 16K of memory.

Table 13.1 Transfer vector offset address

EMT	Transfer Vector	Offset Address (hex)	Comments
GETSYN	GETGAP	0F	COS 3.0 and 3.4, and all ROS versions
LPSTAT	LPSTATV	0F	COS 4.0 and 4.2
KBDTL	KBDTLV	12	
KBDWF	FBDWFV	15	
OUTC	VTV	18	
KBDC	KBDV	1B	
PUTBYT	TOV	1E	In COS 4.0 and 4.2, breaks to Front Panel
GETBYT	TIV	21	
LPOUT	LPV	24	
OUT1	OUT1	27	
OUT2	OUT2	2A	
IN1	IN1	2D	
IN2	IN2	30	
IN3	IN3	33	
-	TRAPX	36	
-	KBDPRE	3E	ROS 1.1, 1.2 and 2.2
LPSTAT	LSTATV	44	COS 3.0 and 3.4; all ROS versions
BOOT	BOOTV	6B	ROS only (breaks to Front Panel in ROS 1.0 & 1.1)
INIT	INITV	6E	
RDSEC	RDSECV	71	In ROS 1.0 and 1.1, breaks to Front Panel
WRSEC	WRSECV	74	
WRCHKV	WRCHKV	77	
INISYS	INISYSV	-2A	COS 4.2 and ROS 1.2
RDSECP	RDSECPV	-27	
WRSECP	WRSECPV	-24	
WRCHKP	WRCHKPV	-21	
RDSECL	RDSECLV	-1E	
WRSECL	WRSECLV	-1B	
WRCHKL	WRCHKLV	-18	
FLUSH	FLUSHV	-15	
RDINFO	RDINFOV	-12	
FORMAT	FORMATV	-0E	
VERTRK	VERTRKV	-0C	
SETLST	SETLSTV	-09	
S4KIN	S4KINV	-06	
S4KTL	S4KTLV	-03	

The base address also varies with each firmware version, but you can find its address using the BASE EMT. The transfer vector address is given by:

Transfer Vector address = Base of defined area of workspace (from EMT BASE) + Offset address (from table 13.1)

Some vectored EMT routines (present in early firmware versions) are not used in later versions, but the EMT can still be called; the transfer vector contains 0FFH, causing a break to Front Panel. GETBYT is an example of this type of EMT. The vector is initialized to break to Front Panel in COS 4.0 and 4.2.

This type of EMT can be used for device handling (in COS 4.0 or 4.2) by changing the address (0FFH) in the transfer vector to the address of your handling routine, as shown in figure 13.3.

When you call GETBYT, the EMT dispatcher jumps to the GETBYT transfer vector (jump C in figure 13.3(a)).

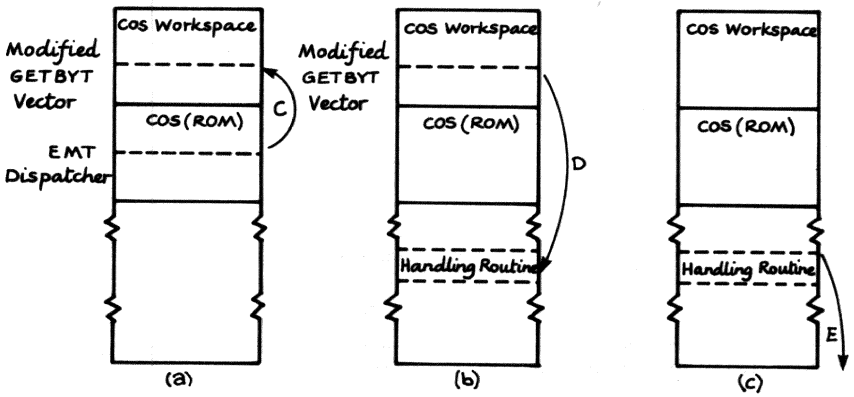


Figure 13.3 Adding a device-handling routine

This vector now contains the address of the start of your handling routine, so jump D (figure 13.3(b)) jumps to it. When the routine has been executed, control returns to your program (jump E in figure 13.3(c)).

The TRAPX Vector

Another way of adding device handlers is to use the TRAPX vector (see table 13.1). Control passes to it if your EMT code number is greater than the highest code number in your Firmware version. It is initialized to break to Front Panel.

If you store a jump instruction at TRAPX:

C3 nnH
(where nnH is the start address of your handler routine)

a jump to your routine will occur.

At this point, register A contains the code number and register HL contains

the address at which it is stored. The previous values of AF and HL are on the stack, therefore these must be popped before a RET instruction can occur, otherwise the contents will be lost. Here is an example program where the vector at TRAPX is assumed to have been initialized earlier to jump to My.emt.handler:

```
My.emt.handler:

        cp      some.number         ;Check that the EMT call is for
                                    ;the EMT which currently interests
        jr      z,enter             ;us and jump to Enter if it is,
        jp      old.trapx vector    ;else jump to old TRAPX vector
                                    ;(this is assumed to have been
                                    ;saved somewhere).

Enter:                              ;Entry point to device servicing
                                    ;code.

        pop     af                  ;Recover working registers.
        pop     hl

                                    ;Then do whatever is needed to
                                    ;service the device.

        ret                         ;Return to calling routine.
```

Note that any program that intercepts the TRAPX vector must restore the old value before exit, or strange results may occur later.

Another example of the use of the TRAPX vector is in simulating the action of EMT VERSN so that your program will run in COS 3.0 (where EMT VERSN is not present):

```
        emt     base                ;This calculates the address of the
        ld      hl,trapx.offset     ;TRAPX vector (by adding its offset
        add     hl,de               ;from the base of COS/ROS workspace
                                    ;to the current value of that base.
        push    hl                  ;The address is left in HL

        ld      de,trapx.save       ;This saves the old contents of the
        ld      bc,3                ;TRAPX vector (note that all 3 bytes
        ldir                        ;must be saved).
        pop     de                  ;
        push    de                  ;The TRAPX vector is now in DE.


        ld      hl,my.trapx.jump    ;A new vector is inserted into TRAPX
        ld      bc,3                ;to point to our new routine.
        ldir

        emt     versn               ;This calls EMT VERSN.  If we are
        pop     de                  ;in a version of COS earlier than
        push    af                  ;3.4 this will result in a call to
                                    ;TRAPX and, thence, to our own
                                    ;routine.

        ld      hl,trapx.save       ;This recovers the old contents of
        ld      bc,3                ;TRAPX.  Although not strictly
        ldir                        ;necessary in this code fragment, it
        jp      rest.of.program     ;is always good practice; failure
                                    ;to do so could result in some
my.trapx.jump:                      ;strange behaviour later on.
        jp      my.trapx

trapx.save:
        defs    3                   ;A place to keep the old contents
                                    ;of TRAPX.

my.trapx:
        pop     af
        pop     hl
        ld      a,30                ;Pretend to be COS 3.0.
        ret
```

The KBDPRE Location

In 480Z machines (ROS 1.1, 1.2 and 2.2), when a keyboard key is pressed, the character generated goes to the keyboard buffer. The character is also placed in a workspace location called KBDPRE (see table 13.1).

Using KBDPRE, languages such as BASIC, LOGO, and PASCAL detect escape characters when they are pressed, rather than waiting for them to go through

the keyboard buffer. After KBDPRE has been read, the character is cleared and replaced with 03FH (the ? character).

If your software uses an EMT to read the keyboard, you should be aware that the keyboard buffer will be cleared but KBDPRE will not. The solution is to include instructions in your machine-language program to clear KBDPRE.

The following example will do this:

```
ld      hl,3eh
emt     base            ;Find base address.
add     hl,de           ;Find KBDPRE address
ld      (hl),'?'        ;Load ? into KBDPRE.
```

Filters

In some instances you may wish to use an EMT only if a certain condition exists. This can be done with EMTs called using a transfer vector.

You can change the contents of the vector to point to your routine as discussed above. However in this case, your routine checks if the condition exists:

- If it does, control is transferred to the EMT routine then back to your program after execution of the EMT.
- If it does not, control returns to your program without execution of the EMT.

IMPORTANT: If you do use this filtering facility, ensure that your routine is placed above 0BFFFH (in common RAM). In the 480Z, EMTs called using a transfer vector switch between memory pages (see chapter 12). Consequently, their vector points to common RAM, an area of memory that is RAM in all pages.

Under CP/M, make sure that your filtering routine is placed above 0BFFFH by doing the following:

- Build a 55K CP/M using MOVCPM
- Copy the 55K image onto your system disc with SYSGEN (S option)
- Press reset, then load the 55K CP/M
- Use DDT to load the .Hex file into memory
- Locate your program at 0C00H upwards.

Under CP/NOS the MOVCPM utility is not available, so you cannot use the above procedure.

The following program uses an OUTC vector filter to change dollar characters sent to the screen to pound characters, by outputting an escape sequence, ESC, '!', 26 in place of the dollar sign:

```
org         0c000h

outc        equ     1                   ;OUTC emt
base        equ     39                  ;BASE emt
warm_boot   equ     0                   ;Warm boot address
esc         equ     27                  ;Escape char
dollar      equ     '$'
pound       equ     26
pling       equ     '!'
                                        ;install_my_outc_routine:
ld          sp,install_my_outc_routine - 2
emt         base                        ;Get pointer to
                                        ; vector base in DE.
ld          hl,.vtv + 1
add         hl,de
push        hl                          ;(hl) points to OUTC
                                        ;vector address.
ld          a,(hl)                      ;Install vector in
                                        ;my_outc_routine.
inc         hl
ld          h,(hl)
ld          l,a
ld          (old_outc_vector + 1),hl
pop         hl                          ;Restore pointer to
                                        ;outc vector.
ld          de,my_outc_routine
ld          (hl),e                      ;Install
                                        ;my_outc_routine
                                        ;pointer in outc
                                        ;vector.
inc         hl
ld          (hl),d
jp          warm_boot
                                        ;New OUTC routine
                                        ;my_outc_routine:
cp          dollar
ld          a,esc                       ;Escape sequence
                                        ;for 'pound sign'
call        old_outc_vector
ld          a, pling
call        old outc vector
ld          a,pound                     ;Graphic char.
                                        ;for 'pound sign'.
old_outc_vector:
jp          0                           ;NB: overlayed by
                                        ;install routine.
```

NOTE: As this program stands, it will convert all dollar characters to pound characters, including those in control character sequences and escape sequences.

CHAPTER 14

POSITION-INDEPENDENT CODE

This chapter describes position-independent code (PIC); this is a form of machine language that is written to produce sections of code that can run at any address.

The first section of this chapter is an introduction to PIC. In the second section there is a description of a firmware instruction, CALR; this is a relative call instruction that is very useful when writing PIC.

INTRODUCTION

It is often convenient to be able to write sections of code that can run at any address at which they are loaded. This type of code is called position-independent code, or PIC.

PIC is useful for sub-routines that are frequently used, since thay can be attached to programs without having to be located at a specific address. PIC is also useful if you want to add a sub-routine to a different system; you can place the sub-routine in any vacant part of the memory. In both cases, PIC can be used directly.

Of course, you could achieve a similar effect by using an assembler to produce a relocatable object module, but this requires relocation before it can be used.

Differences Between PIC and Relocatable Code

A relative jump instruction performs the same function as an absolute jump instruction; both transfer control from one part of a program to another. The difference is the way that the jump destination is calculated:

- an absolute jump collects a given two-byte address from the register containing the address, and loads the address into the program counter.
- a relative jump goes back or forward a specified number of spaces. The specified number is known as the offset number.

The relative-jump type of instruction is necessary for PIC, where data is either relative to the program counter or addressed in the stack.

Relocatable code can be placed anywhere in memory but all addresses are relative to the start of the program. Consequently, relocatable code is address-dependent.

To run relocatable code with another program, you have to use a linker to adjust the addresses in the new program.

THE CALR INSTRUCTION

The Z80 possesses some relative jump instructions, but not all of the types needed for writing PIC. There are some useful "register-indirect" load instructions. For example:

```
ld   a,(hl)
ld   (iy + 3),b
```

These can be used in PIC to access memory if you can set up the registers for addressing.

However, the Z80 does not have a relative call instruction. Consequently, the firmware interprets the CALR instruction (E7 hex) to do this. CALR works in a similar way to the EMT instructions, taking the byte following it as the offset distance to jump.

The CALR instruction can be used wherever the CALL instruction would be used, provided that the target sub-routine is within range. As CALR is an interpreted instruction, it takes longer to execute than a CALL instruction.

In addition, CALR can be used in a PIC program to set up an absolute address in an index register. This is shown in lines 13 to 17 inclusive of the example program below, where "BUF" is the label of the desired target address, and $ is the symbol for the assembler's location counter. Complex programs can be written in PIC, taking little more space than a normal program.

The following program, written entirely in PIC, dumps memory to the line printer, one byte per line; each byte is represented by two hexadecimal digits:

```
        calr    $ + 2               ;Put PC on top of stack.
        ld      de,buf - $
        pop     ix
        add     ix,de               ;IX now points to BUF.
        emt     fstlst              ;Get the boundaries of memory dump.
        inc     hl
Next:   emt     kbdc                ;Check for <CTRL/C>.
        ld      a,0dh
        calr    chout               ;New line.
        ld      a,0ah               ;<LF>.
        calr    chout
        ld      a,(de)              ;Get byte
        calr    outp                ;& output it.
        inc     de                  ;Increment pointer.
        dec     hl                  ;Decrement byte count.
        ld      a,h                 ;Has byte count
        or      l                   ;reached zero?
        jr      nz,next             ;No, jump to carry on.
        emt     0                   ;Yes, so end program.

Outp:   push    hl                  ;save byte.
        push    ix                  ;Count.
        pop     hl                  ;HL now points at BUF.
        emt     byteo               ;Convert A to hex in location BUF.
        dec     hl
        dec     hl                  ;Move HL back to start of BUF.
        ld      a,(hl)              ;Get first ASCII digit.
        calr    chout               ;Print it.
        inc     hl                  ;Move HL to second.
        ld      a,(hl)              ;Get second ASCII digit.
        calr    chout               ;And print it.
        pop     hl                  ;Recover byte count.
        ret

                                    ;Subroutine CHOUT.
                                    ;The output channels through this subroutine
                                    ;(to simplify patching) to send output
                                    ;whither required.
                                    ;
Chout:  emt     lpout               ;Char to line printer.
        ret

Buf:    defs    2                   ;Workspace for BYTEO

kbdc    equ     2
byteo   equ     15h
outc    equ     1
lpout   equ     5
fstlst  equ     37
        end     100h
```

Long-Range Calls

Like the Z80 relative jump instructions, CALR has a call range of -126 to +129. If you need a bigger range, the code below can be used:

```
lcalr:
        push    hl              ;SP initially points
        push    de              ;to
        push    af              ;apparent return address.

        ld      hl,6            ;HL now contains address of
        add     hl,sp           ;normal return address (NRA).
        ld      e,(hl)          ;Put NRA
        inc     hl              ;into
        ld      d,(hl)          ;DE.

        inc     de              ;Add 2 to NRA to obtain
        inc     de              ;real return address (RRA).
        ld      (hl),d          ;Put RRA from DE
        dec     hl              ;back onto stack
        ld      (hl),e          ;in place of NRA.

        ex      de,hl           ;Put RRA into HL and (SP+6) into DE.
        dec     hl              ;Point to last location
        ld      d,(hl)          ;before RRA
        dec     hl              ;containing offset (16 bit).
        ld      e,(hl)          ;Put offset into DE.
        add     hl,de           ;Add it to return address

        pop     af              ;Exchange old HL
        pop     de              ;with required subr.address
        ex      (sp),hl         ;without moving SP.
        ret                     ;Takes you to required subr.
```

To use this code, modify the jump vector at 18H to jump to this piece of code. A code fragment could be:

```
        calr    1$

1$:     pop     de              ;Get address of 1$ into DE.
        ld      hl, lcalr - 1$  ;Get offset of LCALR into HL.
        add     hl, de          ;HL now has absolute address of LCALR.
        ld      (019h),hl       ;Place address in vector for restart 18H.
        ld      a, 0c3h
        ld      (18h),a         ;Put in JP instruction.
                .
                .
        rst     18h             ;Equivalent to
        defw    subroutine - $+2 ;CALL SUBROUTINE.
                .
                .
                .
                .
```

APPENDIX A

QUICK REFERENCE GUIDE OF EMT INSTRUCTIONS

This appendix lists the EMT instructions and summarizes their effects. Those instructions marked with an asterisk are called using a transfer vector (see chapter 13).

The following shorthand conventions are used:

= The register contains the value before call.

<- The value is put into the location by the call.

<= This points to (before call).

OUTC Instructions That Send Characters To The Screen

* OUTC	1(01H)	Output byte in register A to the screen. Screen <- A
OUTCNV	22(16H)	As OUTC, not using transfer vector. Screen <- A
MSG	23(17H)	Output message at (HL) to the screen. HL => First character of message Terminated by 0FFH.
OUTNC	42(2AH)	Output byte in A to screen without losing autopage character (COS versions only). Screen <- A Keyboard latch is not cleared.
CURPOS	59(3BH)	Returns the address of the cursor position (x,y), (not COS 3.0 and 3.4). H <- Line L <- Column

Maintenance Instructions That Modify Screen Effects

GRAFIX 13(0DH) Clear the top 20 lines giving a 4 line scroll

SCROLL 14(0EH) Full screen scroll

WIDTH 52(34H) Change screen width or return present state (not COS 3.0 and 3.4)

COS 4.0 and 4.2

A = Width
0 = 40
1 = 80

A = -1 for present state request
0 <- 0 signals 40
1 <- 1 signals 80

All ROS versions

As above, but also returns current scrolling window

A = -2 for scrolling window request
B <- Top line
C <- Bottom line
D <- Leftmost column
E <- Rightmost column

WINDOW 58(3AH) Define window area of the cursor (not COS 3.0 and 3.4)

B <- Top line
C <- Bottom line
D <- Leftmost column
E <- Rightmost column

Instructions That Access Screen Memory

VTOUT 55(37H) Output character to screen (not COS 3.0 and 3.4)

A = Character
H = Line
L = Column
E = Attribute bit (not ROS)

VTIN	56(38H)	Read character from screen (not COS 3.0 and 3.4) H = Line L = Column A <- Character E <- Attribute bits (not ROS)
VTCLR	57(39H)	Clear specified area of the screen (not COS 3.0 and 3.4) B = Top line C = Bottom line D = Leftmost column E = Rightmost column
VTLINE	60(3AH)	Output line or part of line C = Number of characters in string DE => String H = Line L = Column

Synchronizing Instructions For Screen Memory Addressing

OPNWT	11(0BH)	Open screen for memory access (COS versions only)
CLOSE	12(0CH)	Clear screen (COS versions only)
CLEAR	15(0FH)	Clear selected screen band (COS versions only) HL => Start of top line A = Number of lines to clear HL <- (Last character +1)
OUTF	43(2BH)	Output to screen at (HL) from A register (COS versions only) (HL)<- A
INF	44(2CH)	Input from screen at (HL) to register A (COS versions only) A <- (HL)

Character-Pattern Generating Instructions

CHGEN 53(35H) Generate a new character pattern
(COS 4.0 and 4.2 only)

A = Character
DE => Bit pattern
B = 0 for normal use (wait for frame blanking)
B = 1 for immediate use

CHREAD 54(36H) Read current character pattern
(COS 4.0 and 4.2 only)

A = Character
DE => Bit pattern
B = 0 for normal use (wait for frame blanking)
B = 1 for immediate use

Recommended Keyboard-Handling Instructions

KBDTL 31(1FH) Test keyboard for depression

A <- TRUE (0FFH) if a character
A <- FALSE (00H) if a character
Z set if no character

KBDW 33(21H) Wait for a character, read it, clear the keyboard

A <- Keyboard entry
Trap <CTRL/A>
Blinking active

* KBDWF 34(22H) As KBDW plus test for entry to Front Panel

A <- Keyboard entry
Trap <CTRL/A>
Trap <CTRL/F>
Blinking active

Other Keyboard-Handling Instructions

* KBDC 2(02H) Read the keyboard, trap CTRL/C

A <- Keyboard entry
A <- 0 if no character
Z set if no character
Trap <CTRL/C>

KBDIN 29(1DH) Read and clear the keyboard

A <- Keyboard entry (not <CTRL/A> in ROS)
A <- 0 if no character
Z set if no character

KBDTC 30(1EH) Test keyboard and return character

A <- Keyboard entry
A <- 0 if no character
Z set if no character
Trap <CTRL/A>

Printer And Interface Handling

* LPOUT 5(05H) Output byte in register A to interface

Interface <- A

* OUT1 6(06H) Undefined output EMT linked to I/O channel

* OUT2 7(07H) As OUT1

* IN1 8(08H) Undefined input EMT linked to I/O Channel

* IN2 9(09H) As IN1

* IN3 10(0AH) As IN2

* SETLST 41(29) Set printer (or device) from registers A and E

A = Interface
E = Baud rate

Interface Code

0	=	Screen
1	=	SIO-1 (not ROS)
2	=	SIO-2/2B/3
3	=	User I/O port
4	=	SIO-4
5	=	SIO-5 (not ROS)
6	=	SIO-6 (not ROS)

Baud Rate Code

0	=	110
1	=	300
2	=	600
3	=	1200
4	=	2400
5	=	4800
6	=	9600

* S4KTL	47(2FH)	Test SIO-4 interface for a character A <- 1 if character A <- 0 if no character Z set if no character
* S4KIN	48(30H)	Read the SIO-4 interface into register A A <- Character (minus most significant parity bit)
* LPSTAT	50(32H)	Check that the printer (or device) is ready (not COS 3.0) A <- -1 if ready A <- 0 if not ready Z set if not ready

Cassette-Handling Instructions

* PUTBYT	3(03H)	Output the byte in register A to tape (not COS 4.0 and 4.2) Tape <- A
* GETBYT	4(04H)	Read byte from tape into register A A <- Tape Carry set if control character entered at keyboard
* GETSYN	17(11H)	Get synchronization character from tape (not COS 4.0 and 4.2) Wait until A = Tape Carry set on abortion
SETCAS	40(28H)	Set data transfer speed from register A (not COS 4.0 and 4.2) A = Option A <- Old option
CASCTL	63(3FH)	Initialize the cassette system (ROS versions only) A = 1 before GETBYT A = 2 before PUTBYT A = 0 after a read or write operation

Disc FDC Instruction

The following 4 instructions (INIT, RDSEC, WRSED, WRCHK) use a 5-byte disc parameter block pointed to by register IX:

				FDS	MDS
IX	=>	DEFB	UNIT	(0-3)	(0-3)
		DEFB	TRACK	(0-77	(0-39)
		DEFB	SECTOR	(1-26)	(1-16)
		DEFW	ADDRESS		

ADDRESS is the address of a 128-byte data buffer.

ERROR CODES returned in register A:

BIT	MEANING
7	Disc not ready
6	Disc write-protected
5	Write fault
4	Record not found (addressing error)
3	CRC error (data problems)
2	Data error (hardware)
1	Data mismatch

*	INIT	25(19H)	Initialize disc unit
*	RDSEC	26(1AH)	Read sector
*	WRSEC	27(1BH)	Write sector, no check
*	WRCHK	28(1CH)	Write and check sector
	BOOT	49(31H)	Initiate loading of the disc operating system

Disc IDC Instructions

The instructions, RDSECP, WRSECP, WRCHKP, FORMAT and VERTRK use a 5-byte disc parameter block pointed to by register IX:

				FDS	MDS	
					double	quad
IX	=>	DEFB	UNIT	0-7	0-7	0-7
		DEFB	TRACK	0-77	0-39	0-79

			Single Density	Double Density	Single Density	Double + Quad Density
		DEFB	SECTOR	1-26	1-26	1-16	1-9
		DEFW	ADDRESS				

ADDRESS is the address of a 128, 256, or 512-byte data buffer, depending on the size of the physical or logical sector.

The instructions, RDSECL, WRSECL, and WRCHKL use the same parameters except for SECTOR:

		FDS		MDS	
		Single Density	Double Density	Single Density	Double + quad Density
DEFB	SECTOR	1-26	1-52	1-16	1-36

The instructions INISYS, FLUSH and RDINFO use one control byte pointed to by register IX:

For INISYS

IX => Control Byte

bits 0-2	=	Drive number (0-7) for booting
bit 3	=	Drive change
bits 4-7	=	Number of retries (0-14)

For FLUSH

IX => Control byte

bits 0-2	=	Disc drive (0-7)
bits 3	=	All buffers (1) or only the buffers on specified drive (0)
bits 4-7	=	Ignored

For RDINFO

IX => Control byte

bit 0	=	0 if 5.25 inch 1 if 8 inch
bit 1	=	0 if single-density 1 if double-density
bit 2	=	0 if 48 tpi disc 1 if 96 tpi disc
bit 3	=	0 if 48 tpi drive 1 if 96 tpi drive
bits 4-7	=	0

Error codes are returned in register A:

With Z = 0
Carry = 1 Error while instruction or parameters in transit

Bit	Meaning
0-4	Not used
5	One or more parameters out of range
6	Command unknown to the IDC
7	Drive not ready

With Z = 0 — Disc error
Carry = 0

Bit	Meaning
0	Refers to a previous sector when flushing a buffer
1	Verification error during write/check EMT
2	Not used, defined as 0
3	CRC error
4	Seek error
5	Unknown disc error
6	Disc is write-protected
7	Drive not ready

Successful completion messages are returned in register A (with Z = 1, Carry = indeterminate):

- With INISYS, the number of retries before success is returned
- With RDSECP:

  0 = 128-byte sector
  1 = 256-byte sector
  3 = 512-byte sector

- With RDINFO:

  0 = 0 if 5.25-inch
  1 if 8-inch
  1 = 0 if single density
  1 if double or quad density

* INISYS 64(40H) Initialize disc system

* RDSECP 65(41h) Read physical sector

* WRSECP 66(42H) Write physical sector

* WRCHKP 67(43H) Write and check physical sector

* RDSECL 68(44H) Read logical sector

* WRSECL 69(45H) Write logical sector

* WRCHKL 70(46H) Write and check logical sector

* FLUSH 71(47H) Flush buffer

* RDINFO 72(48H) Read disc drive information

* FORMAT 73(49H) Format floppy disc track

* VERTRK 74(4AH) Verify format of a disc track

Miscellaneous Instructions

ERROR	-1(0FFH)	Print ?Err?, return to firmware command level
CONTC	0(00H)	Return to firmware command level
WAIT	16(10H)	Wait B multiplied by 833 microseconds B = interval time
UPDATE	18(12H)	Copy MASK to PORT0 (COS versions only) Update PORT0 location from MASK location at 0FF03H
GETHEX	19(13H)	Get a hex number from keyboard into HL HL <- Number C <- Number of hex digits B <- Terminating character
DEOUT	20(14H)	Put register DE to (HL) as hex digits HL => Start HL <- (Last character + 1)
BYTEO	21(15H)	Put register A to (HL) as hex digits HL => Start HL <- Last character +1
CHAN	24(18H)	Execute the EMT given in register C C <= EMT code Other registers should be set according to the chosen EMT.
SAVE	35(23H)	Save registers HL, DE and BC Push HL, DE, BC
SAVEA	36(24H)	Save registers A, HL, DE and BC Push HL, DE, BC, AF
FSTLST	37(25H)	Input first and last 16-bit numbers DE <- First BC <- Last HL <- (Last - First)

GETJP 38(26H) Return Ath entry from table at (HL)

A = table entry number
HL => Table
HL <- (HL+A+A)

BASE 39(27H) Return workspace address in register DE

DE <- Firmware RAM workspace

VERSN 51(33H) Return version number of the firmware

A <- COS version * 10
A <- (ROS version * 10) + 100

MOVBLK 61(3DH) Move copy of a block of memory
(ROS 1.0 only)

IX = Number of bytes to be moved
C => Block to be
DE => copied

B <= Area to
HL <= dump at

RAMMAP 62(3EH) Find amount of available memory
(ROS versions only)

In ROS 1.0

B = 0
A <- Number of 16K blocks available

In ROS 1.1, 1.2 and 2.2

As above plus

B = 1
HL <- bit map of available blocks.

APPENDIX B

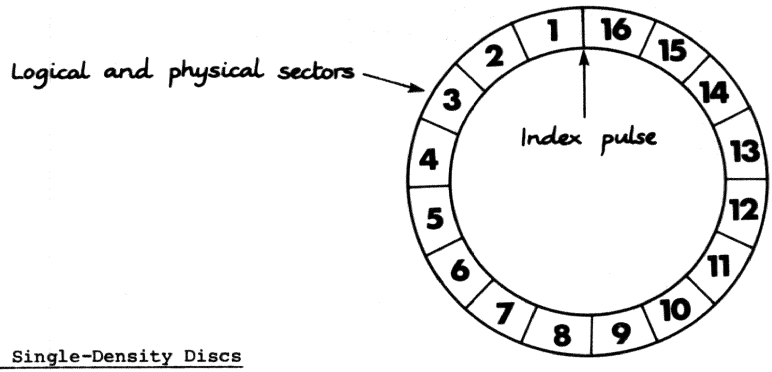
DISC FORMATS

This appendix gives logical formats for each type of disc. In the first section, the IDC logical mapping is shown, and special features are described. In the second section, CP/M logical mapping is shown.

IDC LOGICAL MAPPING

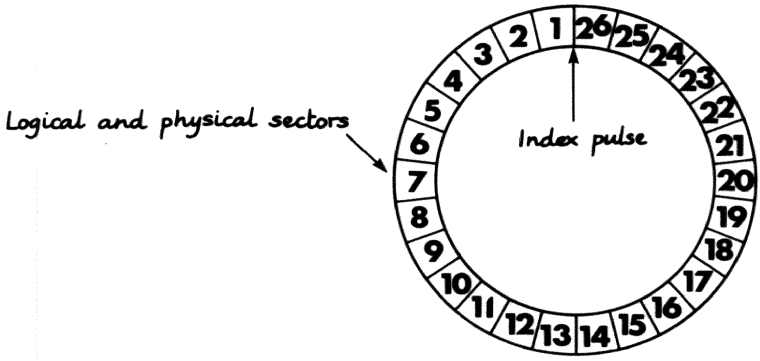
5.25-Inch Single-Density Discs

Here, the logical mapping is exactly the same as the physical format:



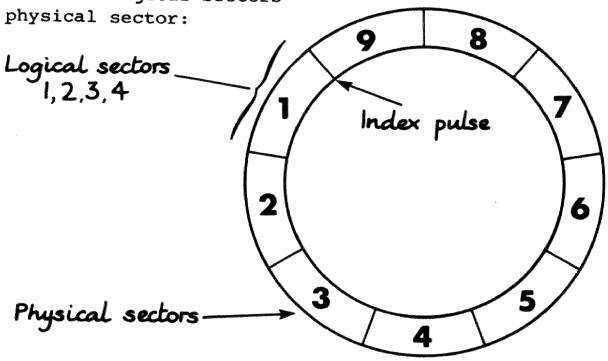
8-Inch Single-Density Discs

Again the logical mapping is the same as the physical format:



5.25-Inch Double and Quad Density Discs

Here, there are 4 logical sectors for each physical sector:



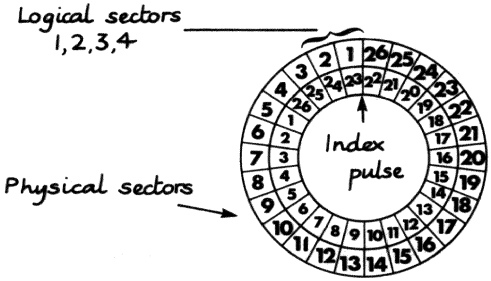
An entire track can be read in 5 revolutions of the disc, assuming that logical sectors are accessed sequentially, as shown below:

Physical sectors	Logical sectors				
	Rev 1	Rev 2	Rev 3	Rev 4	Rev 5
1	1,2,3,4				
2		9,10,11,12			
3			17,18,19,20		
4				25,26,27,28	
5					33,34,35,36
6	5,6,7,8				
7		13,14,15,16			
8			21,22,23,24		
9				29,30,31,32	

On the first revolution, logical sectors 1 to 4 and 5 to 8 are read from physical sectors 1 and 6, respectively. The physical sector reading order is 1, 6, 2, 7, 3, 8, 4, 9, 5.

8-Inch Double-Density Discs

There are 2 logical sectors for each physical sector. Also shown is a "skew" between adjacent tracks. Tracks are skewed by 4 physical sectors:



An entire track can be read in 3 revolutions of the disc, assuming that logical sectors are accessed sequentially, as shown below:

Physical sectors	Logical sectors		
	Rev 1	Rev 2	Rev 3
1 & 2	1,2,3,4		
3 & 4			37,38,39,40
5 & 6		21,22,23,24,	
7 & 8	5,6,7,8		
9 & 10			41,42,43,44
11 & 12		25,26,27,28	
13 & 14	9,10,11,12		
15 & 16			45,46,47,48
17 & 18		29,30,31,32	
19 & 20	13,14,15,16		
21 & 22			49,50,51,52
23 & 24		33,34,35,36	
25 & 26	17,18,19,20		

CP/M LOGICAL FORMATS

As well as the above formats, CP/M imposes its own logical mapping. The layout of this is given below for each type of disc.

5.25-Inch Single-Density Discs

There are 4 logical sectors for each physical sector. The physical sectors are read in the order:

```
 1,  4,  7, 10,
13, 16,  3,  6,
 9, 12, 15,  2,
 5,  8, 11, 14
```

CP/M reserves tracks 0 to 2 for a binary copy of the CP/M system. Track 3 contains the directory and tracks 4 to 39 are data tracks.

Space for files is allocated in 8-sector (1Kbyte) units that are numbered from 0 at the start of track 3.

8-Inch Single-Density Discs

There are 4 logical sectors for each physical sector. The physical sectors are read in the order:

```
 1,  4, 13, 19,
25,  5, 11, 17,
23,  3,  9, 15,
21,  2,  8, 14,
20, 26,  6, 12,
18, 24,  4, 10,
16, 22
```

CP/M reserves tracks 0 and 1 for a binary copy of the CP/M system. Track 2 contains 16 sectors of directory followed by 10 data sectors. Tracks 3 to 76 are data tracks.

Space for files is allocated in 8-sector (1Kbyte) units that are numbered from 0 at the start of track 2.

5.25-Inch Double and Quad-Density Discs

CP/M reserves tracks 0 to 2 for a binary copy of the CP/M system. Track 3 contains 16 logical sectors of directory followed by 20 data sectors. Tracks 4 to 39 are data tracks.

Note that track 0 is in single-density format; all other tracks are in double-density format.

Space for files is allocated in 8 logical-sector (1Kbyte) units that are numbered from 0.

8-Inch Double-Density Discs

CP/M reserves tracks 0 to 1 for a binary copy of the CP/M system. Track 2 contains 32 directory sectors followed by 20 data sectors. Tracks 3 to 76 are data tracks.

Note that track 0 is in single-density format; all other tracks are in double-density format.

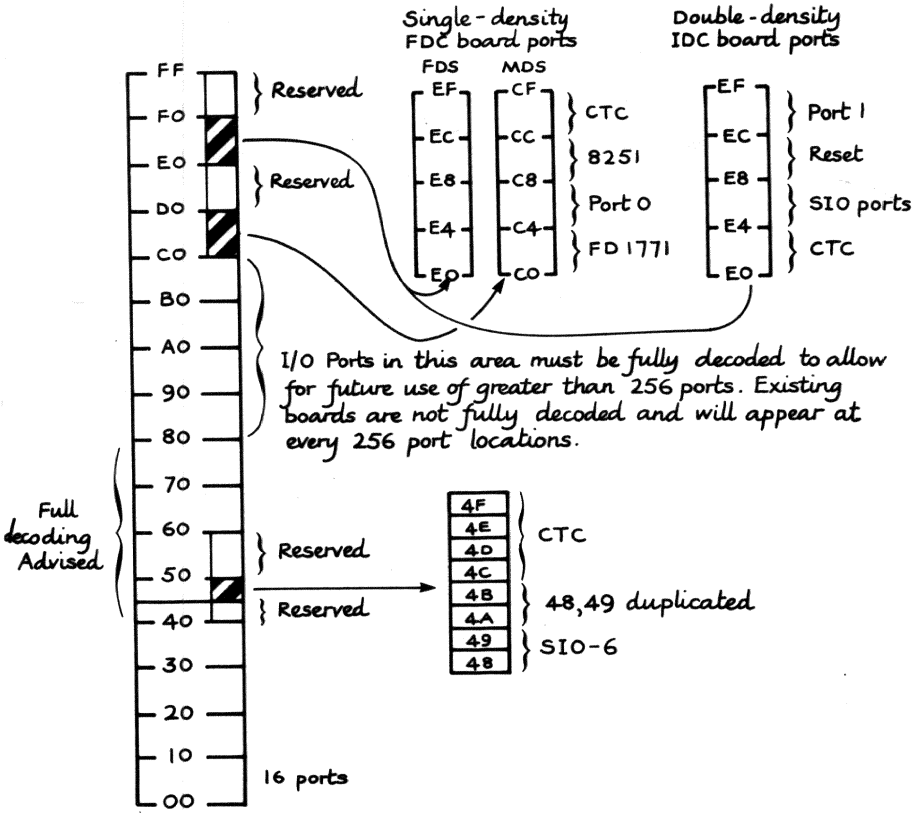
Space for files is allocated in 16 logical-sector (2 Kbyte) units that are numbered from 0.

APPENDIX C

I/O PORTS

380Z I/O PORTS

Figure C.1 shows the 380Z I/O ports.



Note: PIO boards occupy 16 consecutive I/O ports.
PIO + IEEE board occupies 32 consecutive I/O ports.
Users are advised to fill I/O ports from the bottom avoiding 40 to 5F if possible.

Figure C.1 380Z I/O Ports

Details of the IDC board ports (COS 4.2 only) are given below:

Address			
E0	:	CTC channel 0 (Channel B Rx/Tx)	- read/write
E1	:	CTC channel 1 (Channel A Tx)	- read/write
E2	:	CTC channel 2 (Channel A Rx)	- read/write
E3	:	CTC channel 3 (Completion Interrupt)	- read/write
E4	:	SIO channel 'A' data	- read/write
E5	:	SIO channel 'A' control	- read/write
E6	:	SIO channel 'B' data	- read/write
E7	:	SIO channel 'B' control	- read/write
E8	:	IDC interface status (Port 2)	- read only
E8	:	IDC reset	- write only
E9	:	IDC reset	- write only
EA	:	IDC reset	- write only
EB	:	IDC reset	- write only
EC	:	IDC interface data/commands (Port 1)	- read/write
ED	:	IDC interface data/commands (Port 1)	- read/write
EE	:	IDC interface data/commands (Port 1)	- read/write
EF	:	IDC interface data/commands (Port 1)	- read/write

equivalent of Appendix C of Firmware Ref. Manual for the FDC.

ie Details of the FDC board ports for Cos 4.0 and 3.4

480Z I/O PORTS

I/O Ports for the 480Z are listed below:

Port Address	Function		Options needed
0-17H	VDU Port 0 corresponds to top line of screen Port 17H corresponds to bottom line of screen		
18H	Control / Status Port 0		
19H	Control / Status Port 1		
1AH	Control / Status Port 2		
1BH	Control / Status Port 3 (DAC)		e
1DH	Control / Status Port 5 (USERIO Port)		
20H	Main Board CTC channel 0 - SIO-4 and cassette input		
21H	Main Board CTC channel 1 - Cassette I/O and SIO-2		
22H	Main Board CTC channel 2 - Keyboard interrupts		
23H	Main Board CTC Channel 3 - 50Hz interrupts (for repeat key)		
24H	SIO channel A (Network) data port		
25H	SIO channel B (SIO-4) data port		
26H	SIO channel A (Network) control / status port		
27H	SIO channel B (SIO-4) control / status port		
28H	Maths chip Data port		ab
29H	Maths chip Control / Status port		ab
2AH	(Maths chip data port)		ab
2BH	(Maths chip Control / Status port)		ab
2CH	Option Board CTC channel 0 - IEEE interrupts		acd
2DH	Option Board CTC channel 1 - Maths chip interrupts		abd
2EH	Option Board CTC channel 2 - Real Time Clock		ad
2FH	Option Board CTC channel 3 - Real Time Clock		ad
30H	IEEE 0 Interrupt Status 0	Interrupt Mask 0	ac
31H	IEEE 1 Interrupt Status 1	Interrupt Mask 1	ac
32H	IEEE 2 Address Status		
33H	IEEE 3 Bus Status	Auxiliary Command	ac
34H	IEEE 4	Address register	ac
35H	IEEE 5	Serial Poll register	ac
36H	IEEE 6 Command Pass Thru	Parallel Poll register	ac
37H	IEEE 7 Data Input	Data Output	ac
38H	HRGPORT 0 Dil Switch	Y Address Look-up data	a
39H	HRGPORT 1	X Address	a
3AH	HRGPORT 2 HRG Status	Control Look-up address	a
3BH	HRGPORT 3 Data Input	Data Output	a

Options

a) Option board
b) Maths chip
c) IEEE chip
d) CTC chip
e) DAC

Further information can be found in the 480Z Information File.

INDEX

Where there is more than one entry, the first is the most significant.

USERS' COMMENTS

To help Research Machines produce the highest quality microcomputers, supporting software and technical publications, we like to hear from users about their experiences with our products.

Do share your thoughts with us by jotting them down on the tear-off form on the next page. You can leave out your personal details, if you want to. Fold the form in two, seal it with adhesive tape and put it in the post. No stamp is needed if you post it within the United Kingdom.

If you would like to give more information than we have allowed room for on the form, we will be very pleased to receive a letter from you. You can even use the form to ask for a post-paid envelope if you wish.

Additional information will be most useful. If you give us as much detail as possible about your hardware configuration, software version number or manual title, then we can relate your comments to the correct product.

Seal with self-adhesive tape (not staples) along this edge.

RESEARCH MACHINES

MICROCOMPUTER SYSTEMS

Fold along this line.

Postage will be paid by licensee

Do not affix Postage Stamps if posted in Gt Britain, Channel Islands, N Ireland or the Isle of Man

1

BUSINESS REPLY SERVICE
Licence No OF32.

TECHNICAL PUBLICATIONS DEPT
RESEARCH MACHINES LTD
PO BOX 75 OXFORD
OX2 0BR

USER'S COMMENT FORM

PN 10971

User's comments help us to improve our products. If you would like to make any comments, please use this reply-paid form.

Your comments:

Research Machines may use this information in any way believed to be appropriate and without obligation.

Although it is not essential, it would be helpful if you gave the following information:

Name .....

Organization .....

Address .....

..... Post Code .....

System: 380Z / 480Z / Network ☐ Cassette / 5.25" discs / 8" discs
(Delete as necessary)